o'reillys basics

2. Auflage

Praxiswissen Flash 8

→ Der praxisnahe Einstieg in Flash Professional 8 und Flash Basic 8

→ Konkrete Anleitungen und aussagekräftige Beispiele

→ Mit Beispielprojekt auf CD-ROM



O'REILLY®

Sascha Kersken

2. AUFLAGE

Praxiswissen Flash 8

Sascha Kersken



Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
Tel.: 0221/9731600
Fax: 0221/9731608
E-Mail: kommentar@oreilly.de

1. Auflage 2005
2. Auflage 2006

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten
sind im Internet über *http://dnb.ddb.de* abrufbar.

Lektorat: Inken Kiupel, Köln
Fachgutachten: Ralf Bokelberg, Köln
Korrektorat: Petra Bromand, Düsseldorf
Satz: Tim Mergemeier, reemers publishing services gmbh, Krefeld; www.reemers.de
Umschlaggestaltung: Hanna Dyer & Ellie Volckhausen, Boston
Produktion: Andrea Miß, Köln
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell; www.koeselbuch.de

ISBN-10 3-89721-450-4
ISBN-13 978-3-89721-450-7

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

In diesem Kapitel:

- ActionScript im Überblick
- Ein erstes Beispiel
- Die Aktionen-Palette
- ActionScript-Grundelemente
- Movieclips steuern
- Praxisbeispiel: Navigation, interaktive Demo und Spiel

KAPITEL 6

Interaktivität durch ActionScript

Bender: Ich hatte einen furchtbaren Traum:
Einsen und Nullen überall. Und ich dachte, ich hätte 'ne Zwei gesehen!

Fry: Ach Bender, das war doch nur ein Traum.
So was wie Zweien gibt es gar nicht!

– Dialog aus *Futurama*

In den bisherigen Kapiteln haben Sie einen guten Überblick über die »Designer«-Aspekte von Flash erhalten und mächtige Werkzeuge zum Zeichnen sowie für die Animation und Medienintegration kennen gelernt. All diese Hilfsmittel machen Flash zur ersten Wahl für multimediales Webdesign. Dennoch stellen sie nur die eine Hälfte seiner Fähigkeiten dar: Erst die eingebaute Programmiersprache *ActionScript* ermöglicht das Erstellen interaktiver Webanwendungen mit Flash. In diesem Kapitel lernen Sie die wichtigsten Grundzüge der Steuerung von Flash-Filmen und ihren Inhalten durch ActionScript kennen.



Auf der CD zum Buch finden Sie den speziellen Ordner */actionscript*, der zu allen Themen aus dem Theorieteil dieses Kapitels komplette Beispielfilme enthält. Darüber hinaus enthält er ein Unterverzeichnis namens *zusatz* mit Beispielen und Dokumenten zu fortgeschrittenen ActionScript-Themen, die platztechnisch und konzeptionell nicht in dieses Buch passen.

ActionScript im Überblick

Bis zur Flash-Version 3.0 gab es ActionScript noch nicht. Schlüsselbilder und Schaltflächen ließen sich lediglich mit einigen vorgefertigten »Aktionen« aus einem Drop-down-Menü versehen. Diese ermöglichten nicht viel mehr, als den Film anzuhalten und weiterzuspielen, zu einem angegebenen Bild zu springen oder einen anderen Flash-Film beziehungsweise eine andere HTML-Datei in den Browser zu

laden. Die Interaktivität von Flash ging mit anderen Worten nicht sehr weit über das hinaus, was sich mit bloßen HTML-Hyperlinks bewerkstelligen lässt.

Auch in Version 4 wurden die Aktionen noch nicht als ActionScript bezeichnet, und die »Eingabe« war weiterhin nur per Menü möglich. Dennoch brachte bereits diese Version die entscheidenden Neuerungen, die Flash zu einer programmierbaren Umgebung machten: Variablen, dynamische Ausdrücke, Fallentscheidungen und Schleifen.

Erst in Version 5 wurde ActionScript dann formal als echte Programmiersprache eingeführt. Um möglichst vielen Web-Entwicklern einen leichten Einstieg zu ermöglichen, wurde ActionScript ECMA-262-kompatibel gemacht und basierte somit auf demselben Standard der *European Computer Manufacturers' Association* wie auch die verbreitete Browser-Skriptsprache JavaScript. Damit besaß ActionScript eine Syntaxgrundlage, die bereits bei ihrer Einführung weithin bekannt war.

In Flash MX erfuhr ActionScript eine recht umfangreiche Erweiterung durch neue Funktionen. Zwei prinzipielle Neuerungen wurden dagegen erst in Flash MX 2004 eingeführt: Basierend auf der *3rd Edition* des ECMA-262-Standards[1] wurden feste Datentypen sowie vollwertige Objektorientierung durch echte Klassen eingeführt. Diese Erweiterungen rechtfertigen die neue Bezeichnung ActionScript 2.0.



Falls Ihnen Begriffe wie Datentypen oder Objektorientierung nichts sagen: keine Sorge. Weiter unten in diesem Kapitel erfahren Sie Genaueres darüber.

In Flash 8 wurde ActionScript 2.0 nicht wieder grundsätzlich umgebaut, aber um diverse Objekte und Funktionen erweitert. Diese dienen unter anderem der Skriptsteuerung der neuen Fähigkeiten wie Mischmodi und Filter. Inzwischen wird bei Macromedia an ActionScript 3.0 gearbeitet; diese Erweiterung der Sprache wird wahrscheinlich in die nächste Flash-Version eingebaut.

ActionScript ist eine vollständige Programmiersprache, in der sich sämtliche computerlösbaren Probleme (Algorithmen) implementieren lassen.[2] Dennoch sind die Elemente der Sprache stark an die Besonderheiten von Flash angepasst, so dass Sie den Einstieg leicht finden dürften, wenn Sie sich ein wenig mit den sonstigen Funktionen des Programms auskennen – zum Beispiel nach dem Durcharbeiten der bisherigen Kapitel dieses Buches.

Sie können ActionScript-Anweisungen an drei verschiedenen Stellen in einem Flash-Film einsetzen: in Schlüsselbildern, bei Schaltflächen-Instanzen und bei Movieclip-Instanzen. Wenn Sie einem Schlüsselbild (egal, ob im Hauptfilm oder

---

1 Die Spezifikation ist sehr theoretisch und trocken – aber wenn Sie möchten, können Sie sie online unter *http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf* lesen.

2 Nach dem britischen Mathematiker, Kryptologen und Informatik-Pionier Alan Turing (1912–1954) werden solche Sprachen als *Turing-vollständig* bezeichnet.

innerhalb eines Movieclip-Symbols) ActionScript-Code zuweisen, wird dieser ausgeführt, sobald das Schlüsselbild abgespielt wird. Das ist praktisch, wenn eine Aktion automatisch zu einem bestimmten Zeitpunkt während des Filmablaufs stattfinden soll – beispielsweise, wenn Sie den Film auf einem bestimmten Frame anhalten möchten. Wenn Sie einer Schaltflächen-Instanz Code zuweisen, bestimmt dieser, was bei einem Klick auf die Schaltfläche geschehen soll. Movieclip-Instanzen reagieren dagegen auf diverse andere Ereignisse – zum Beispiel können Sie ein Skript schreiben, das immer wieder ausgeführt wird, wenn Sie das Ereignis `enterFrame` verarbeiten, oder über das Ereignis `load` eine einmalige Aktion beim Auftreten der Instanz in Gang setzen. Der vierte mögliche Speicherort für ActionScript befindet sich außerhalb der Flash-Filme selbst: ActionScript 2.0-Klassen müssen in externen *.as*-Dateien gespeichert werden; dies wird im nächsten Kapitel besprochen.

Ein erstes Beispiel

Natürlich endet auch dieses Kapitel wieder mit einem Praxisteil, der die Site des FlashRock Music Shop weiterführt. Allerdings lassen sich auch die Grundlagen einer Programmiersprache am besten anhand von Beispielen vermitteln, so dass Sie in diesem Kapitel auch einige unabhängige Einzelbeispiele finden.

Das allererste einfache Beispiel zeigt, wie ActionScript den linearen Ablauf der Zeitleiste unterbricht und den Film interaktiv macht: Eine Animation wird weder endlos wiederholt, noch wird sie nach einem Durchlauf gestoppt. Stattdessen wird sie genau dreimal abgespielt; anschließend werden zwei Schaltflächen angezeigt, die entweder drei neue Durchläufe starten oder zu einem Abschlussbild springen, in dem der Film anhält.

Schematisch lässt sich der Ablauf des Beispiels also durch das Flussdiagramm in Abbildung 6-1 darstellen. In einem solchen Diagramm stehen rechteckige Felder für einen linearen Ablauf, während rautenförmige Kästen jeweis eine Fallentscheidung bezeichnen.

Erstellen Sie in einem neuen Dokument zunächst drei Ebenen namens *animation*, *buttons* und *action*. Fügen Sie auf der Ebene *action* in Bild 2 ein Schlüsselbild ein – einige ältere Flash Player-Versionen führen im allerersten Frame eines Films kein ActionScript aus. Achten Sie darauf, dass das neue Schlüsselbild in der Zeitleiste noch markiert ist, und öffnen Sie dann das Bedienfeld *Aktionen* (*Fenster → Aktionen* oder **F9**). Deaktivieren Sie hier gegebenenfalls die *Skripthilfe*, damit Sie ActionScript per Hand editieren können, und geben Sie dann folgenden Code ein:

```
var turns = 0;
```

Diese Anweisung erstellt eine Variable (einen benannten Speicherplatz) mit dem Namen `turns` und speichert den Wert 0 darin. Diese Variable wird benutzt, um die drei Animationsdurchgänge zu zählen.

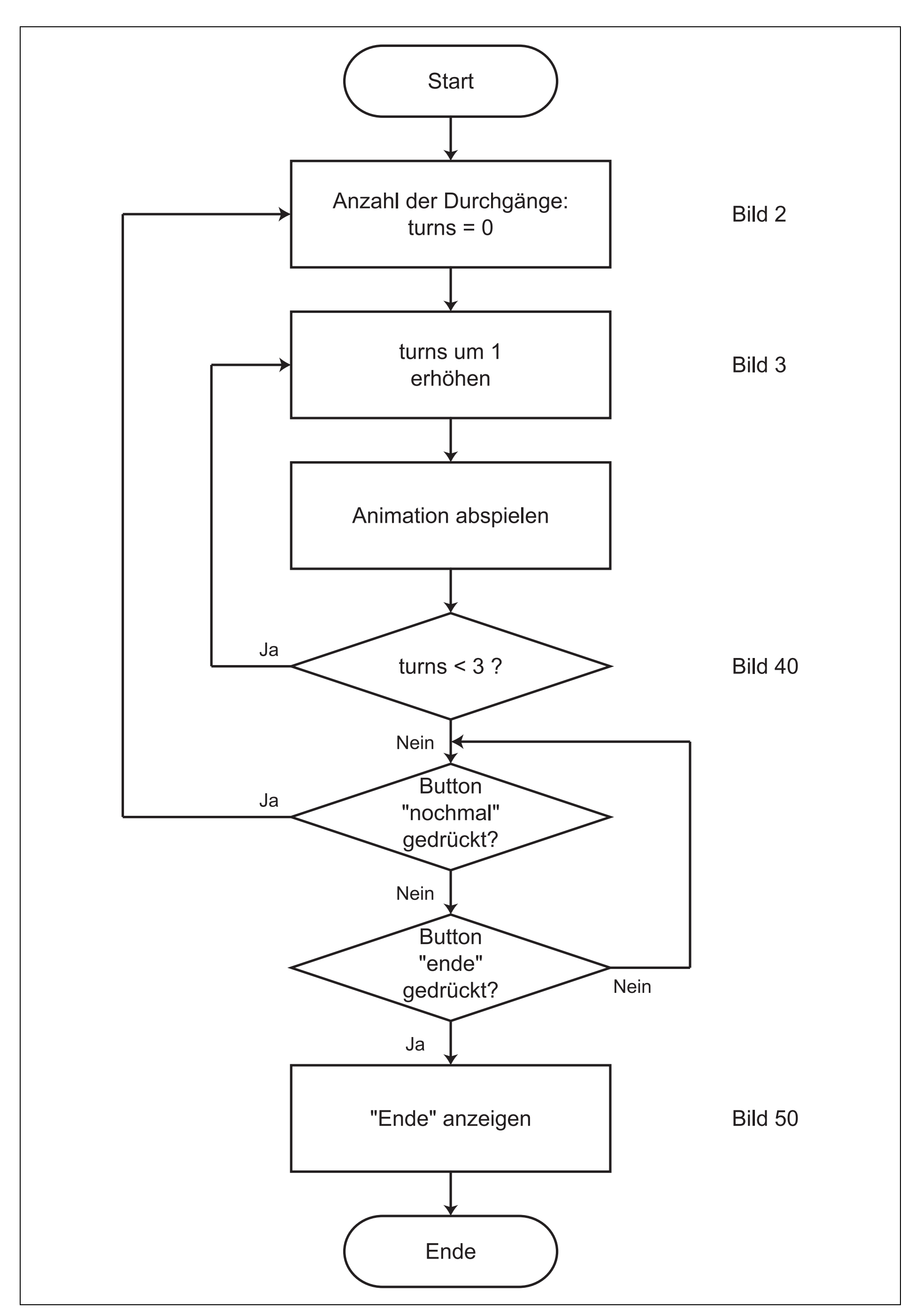


Abbildung 6-1: Das Flussdiagramm erläutert den Ablauf des ersten Beispiels

Ein kleines *a* am Schlüsselbildsymbol in der Zeitleiste weist darauf hin, dass das Schlüsselbild ActionScript-Anweisungen enthält.

Erstellen Sie in Bild 3 auf der Ebene *action* ein weiteres Schlüsselbild. Weisen Sie ihm folgende ActionScript-Anweisung zu:

```
turns++;
```

Dies erhöht den Wert von `turns` um 1 – genau diese bei jedem Durchlauf ausgeführte Anweisung macht `turns` zur Zählervariablen.

Erstellen Sie auf der Ebene *animation* eine beliebige Animationssequenz, die von Frame 3 bis 40 reicht. Eine einfache Tweening-Animation genügt.

Fügen Sie auf der Ebene *action* in Frame 40 ein Schlüsselbild ein. Erstellen Sie im Bedienfeld *Aktionen* folgende Anweisungen für dieses Frame:

```
if (turns < 3) {
   gotoAndPlay (3);
} else {
   stop();
}
```

Diese Anweisungsfolge ist eine Fallentscheidung: Wenn (`if`) der Wert von `turns` noch kleiner als 3 ist, springt der Film zu Frame 3. Dort beginnt die Animationssequenz von Neuem, und `turns` wird wieder um 1 erhöht. Die Funktion `gotoAndPlay()` springt zum angegebenen Bild und spielt die Zeitleiste von dort aus ab. Wenn `turns` dagegen *nicht* kleiner als 3 ist (`else`), hält der Film an der aktuellen Stelle an. Für Letzteres ist die Funktion `stop()` zuständig.

Als Nächstes benötigen Sie zwei Schaltflächen: Wählen Sie jeweils *Einfügen → Neues Symbol* oder drücken Sie **STRG + F8**, um diese Symbole zu erstellen. Geben Sie den Symbolen die Namen *nochmal* und *ende*. Stellen Sie unter *Verhalten* den Typ *Schaltfläche* ein. Zeichnen Sie im Bild *Up* dieser Schaltflächen beliebige Formen, und beschriften Sie sie mit *noch mal* beziehungsweise *Ende*. Erstellen Sie auf der Ebene *buttons* in Frame 40 ein Schlüsselbild. Ziehen Sie dort die beiden Buttons aus der Bibliothek auf die Bühne. Klicken Sie zunächst die Instanz von *nochmal* an, und öffnen Sie dann die Aktionen-Palette. So können Sie folgende Anweisungen für diesen Button erstellen:

```
on (release) {
   gotoAndPlay (2);
}
```

Bei Schaltflächen müssen die Anweisungen in einem Block nach folgendem Schema stehen:

```
on (Ereignis) {
   ...
}
```

Wenn ein Benutzer die Schaltfläche auf die angegebene Weise betätigt, werden die Anweisungen ausgeführt. `on(release)` ist die häufigste Ereignisprozedur – sie wird nach einem Klick und dem anschließenden Loslassen der Maustaste auf der Schaltfläche ausgelöst. Die Bedeutung von `gotoAndPlay()` wurde bereits angesprochen. Beachten Sie, dass der Sprung diesmal zu Frame 2 erfolgt. Wie Sie vielleicht bemerkt haben, steht in diesem Bild das Skript, das den Wert von `turns` auf 0 setzt. Auf diese Weise werden also drei neue Durchgänge eingeleitet.

Die Schaltfläche *ende* erhält folgende ActionScript-Anweisung:

```
on (release) {
   gotoAndStop (50);
}
```

Die Funktion `gotoAndStop()` springt genau wie `gotoAndPlay()` zum angegebenen Bild eines Flash-Films. Allerdings wird der Film nicht von dort abgespielt, sondern hält an. Im vorliegenden Beispiel soll nämlich nur noch ein statisches »Ende«-Bild angezeigt werden, wenn der entsprechende Button angeklickt wurde.

Fügen Sie nun in Frame 50 auf der Ebene *animation* ein *leeres* Schlüsselbild ein (*Einfügen → Zeitleiste → Leeres Schlüsselbild* oder Taste **F7**). Schreiben Sie dann den Text »Ende« auf die leere Bühne. Wenn Sie möchten, können Sie auch auf der Ebene *action* ein Schlüsselbild einfügen und ihm folgende einfache Anweisung zuordnen:

```
stop();
```

Da dieses Frame durch `gotoAndStop()` erreicht wird, ist dies eigentlich unnötig, denn der Film hält ohnehin an. Allerdings wird das durch diese Bild-Aktion deutlicher.

Nachdem Sie alles erledigt haben, können Sie den Film mit *Steuerung → Film testen* oder **STRG + ENTER** starten. Wenn Sie alles richtig gemacht haben, sollte nun dreimal die Animation abgespielt werden. Anschließend müssten die beiden Schaltflächen erscheinen. Wenn Sie *Noch mal* anklicken, sollte die Animation erneut dreimal abgespielt werden, klicken Sie dagegen auf *Ende*, dann müsste das entsprechende Standbild erscheinen.

Tabelle 6-1 zeigt noch einmal zusammenfassend alle ActionScript-Anweisungen, die Sie in diesem kurzen Beispiel benutzt haben. Weiter unten in diesem Kapitel werden sie genauer beschrieben.

Tabelle 6-1: ActionScript-Anweisungen des ersten Beispiels im Überblick

Position	Anweisung	Erläuterung
Bild 2	`var turns = 0;`	Variable `turns` deklarieren und ihr den Wert 0 zuweisen
Bild 3	`turns++;`	Wert von `turns` um 1 erhöhen

Bild 40	`if (turns < 3) {`
	`gotoAndPlay (3);`
	`} else {`
	`stop();`
	`}`
		Fallentscheidung: Wenn `turns` noch kleiner als 3 ist, bei Bild 3 weiterspielen, sonst anhalten

Tabelle 6-1: *ActionScript-Anweisungen des ersten Beispiels im Überblick (Fortsetzung)*

Position	Anweisung	Erläuterung

Button *nochmal*	`on (release) {`
	`   gotoAndStop (2);`
	`}`
		Nach Klick und Loslassen zu Bild 2 springen

Button *ende*	`on (release) {`
	`   gotoAndStop (50);`
	`}`
		Nach Klick und Loslassen zu Bild 50 springen

Bild 50	`stop();`	Film anhalten (optional, da Sprung hierhin mit `gotoAndStop()` erfolgt und bereits stoppt)

Die Aktionen-Palette

Im obigen Beispiel wurde das Bedienfeld *Aktionen* einfach ohne nähere Erläuterung verwendet. Dabei besitzt es einige interessante Eigenschaften; die wichtigsten von ihnen sollen hier kurz angesprochen werden. Einige erweiterte Fähigkeiten, die dem Debugging (der Fehlersuche) in längeren Skripten dienen, gehen dagegen über den Rahmen dieses Buches hinaus. Abbildung 6-2 zeigt das Bedienfeld zunächst einmal im Überblick.



Abbildung 6-2: Die Aktionen-Palette im Überblick

Elemente der Aktionen-Palette

Das Fenster ist in drei Hauptbereiche unterteilt: Links oben können Sie aus einer Baumstruktur sämtliche Funktionen, Objekte und Operationen auswählen, die ActionScript zu bieten hat. Sie können sie durch Ziehen oder per Doppelklick, also ohne Tipparbeit, in Ihre Skripte einfügen.

Übrigens gibt es eine noch schnellere Möglichkeit, häufige Anweisungen in ein Skript einzufügen: Das Bedienfeld akzeptiert zahlreiche *Esc-Tastenkombinationen*. Dafür müssen Sie zuerst die **Esc**-Taste drücken und anschließend zwei Buchstaben eintippen. Zum Beispiel fügt **Esc**, **G**, **P** die Anweisung `gotoAndPlay()` ein; **Esc**, **S**, **T** erzeugt die Funktion `stop()`. Wenn Sie diese Kürzel erlernen möchten, sollten Sie im dem Optionsmenü des Aktionen-Bedienfelds ganz rechts oben die Einstellung *Esc Shortcut Keys* aktivieren.

Links unten finden Sie eine ebenfalls hierarchisch unterteilte Übersicht über sämtliche Skripte, die Bildern oder Objekten des aktuellen Films zugewiesen wurden. Hier können Sie durch einfaches Klicken zwischen den verschiedenen Skripten des Dokuments hin- und herspringen. Eine ähnliche Aufgabe erfüllt übrigens der *Film-Explorer* (*Fenster → Film-Explorer* oder **ALT + F3**); er zeigt aber nicht nur die Skripte, sondern auch andere Objekte des Dokuments in einer hierarchischen Übersicht an.

Den Hauptbereich der Aktionen-Palette bildet natürlich der rechte Fensterteil: Hier werden die eigentlichen Skripte eingegeben. Die Eingabe bleibt besonders durch zwei Aspekte besonders übersichtlich: das *Syntax-Highlighting*, also die unterschiedliche Einfärbung verschiedener Skriptbestandteile, sowie die automatische Einrückung verschachtelter Anweisungen.

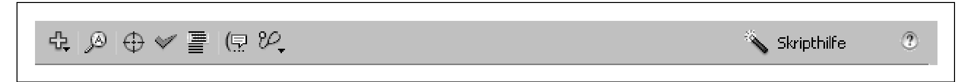


Abbildung 6-3: Die Symbolleiste der Aktionen-Palette

Über dem eigentlichen Skriptbereich finden Sie die Symbolleiste, die in Abbildung 6-3 zu sehen ist. Sie bietet eine Reihe nützlicher Werkzeuge zur Skriptbearbeitung. Von links nach rechts handelt es sich kurz gefasst um folgende Funktionen:

- *Skriptobjekt hinzufügen:* Dieses Pop-up-Menü enthält dieselbe hierarchische Anordnung von ActionScript-Elementen zum schnellen Einfügen wie der linke obere Fensterbereich.
- *Suchen* (Tastenkombination **STRG + F**): Sucht im aktuellen Skript nach dem eingegebenen Text und ersetzt ihn gegebenenfalls durch einen anderen. Zum Weitersuchen nach demselben Suchbegriff können Sie jeweils **F3** drücken. Wenn Sie in *allen* Skripten (und anderen Elementen) des Films nach einem bestimmten Text suchen oder diesen ersetzen möchten, müssen Sie stattdessen den Menübefehl *Bearbeiten → Suchen und ersetzen* (oder **STRG + F** außerhalb des Skriptfensters) verwenden.
- *Zielpfad einfügen:* Ermöglicht die Auswahl einer Symbolinstanz, die Sie im Skript ansprechen möchten, aus einer Baumhierarchie.
- *Syntax überprüfen* (**STRG + T**): Sucht nach schwerwiegenden Fehlern im Skript und zeigt diese an.



Syntaxfehler sind relativ leicht zu finden, denn sie führen dazu, dass ein Skript gar nicht erst ausgeführt werden kann. Formal korrekte, aber logisch fehlerhafte Skripte werden dagegen erst einmal ausgeführt, verhalten sich aber anders als erwartet. Solche Fehler sind nur sehr schwer zu entdecken. Weiter unten wird die nützliche Funktion `trace()` erläutert, die Ihnen dabei helfen kann.

- *Auto-Format* (**STRG** + **Umschalt** + **F**): Formatiert das aktuelle Skript sauber neu. Die Regeln für die Formatierung können Sie unter *Bearbeiten* → *Voreinstellungen* unter der Kategorie *Auto-Format* festlegen.
- *Codehinweis zeigen* (**STRG** + **Leertaste**): In der Regel erscheinen bereits während der Skripteingabe kleine Marken, die die Syntax von Funktionen erläutern (siehe Abbildung 6-4). Mit dieser Schaltfläche können Sie sich diese Hinweise erneut anzeigen lassen.

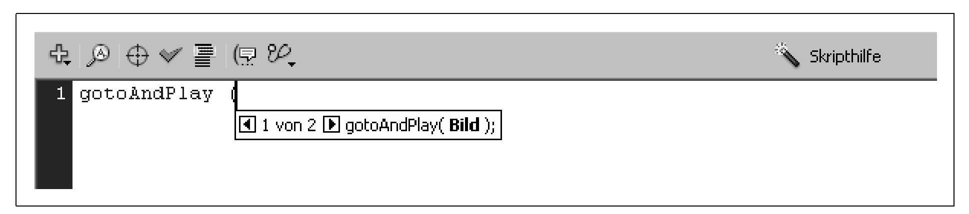


Abbildung 6-4: Codehinweise helfen Ihnen, Funktionen mit korrekter Syntax zu benutzen

- *Debug-Optionen:* Hier können Sie *Haltepunkte* setzen, um die Ausführung des ActionScript-Codes an bestimmten Stellen zu kontrollieren.
- *Skripthilfe* (**STRG** + **Umschalt** + **E**): Aktiviert das dialoggesteuerte Hinzufügen von ActionScript-Elementen (siehe unten).

Das Optionsmenü des Bedienfelds (ganz rechts oben) verfügt über weitere interessante Optionen. Soweit sie nicht bereits als Elemente der Symbolleiste beschrieben wurden, sind dies folgende:

- *Gehe zu Zeile* (**STRG** + **G**) springt sofort zur eingegebenen Zeilennummer.
- *Skript importieren* (**STRG** + **Umschalt** + **I**) importiert ActionScript-Code für den aktuellen Kontext aus einer externen *.as*-Datei.
- *Skript exportieren* (**STRG** + **Umschalt** + **X**) speichert den Code aus dem aktuellen Fenster als externe ActionScript-Datei.
- *Esc Shortcut Keys* aktiviert beziehungsweise deaktiviert die Ansicht der bereits beschriebenen Befehlskürzel mit der **Esc**-Taste.
- *Versteckte Zeichen* (**STRG** + **Umschalt** + **8**) zeigt Leerzeichen, Zeilenumbrüche, Tabulatoren und so weiter durch Hilfssonderzeichen an.

- *Zeilennummern* (**STRG** + **Umschalt** + **L**) schaltet die praktische Anzeige der Skript-Zeilennummern ein beziehungsweise aus.
- *Zeilenumbruch* (**STRG** + **Umschalt** + **W**) sorgt dafür, dass zu lange Skriptzeilen im Fenster umbrochen werden.

Die Skripthilfe

Für weniger erfahrene Flash-Benutzer bietet die *Skripthilfe* die Möglichkeit, die Parameter von ActionScript-Anweisungen im Dialogbetrieb auszuwählen statt manuell einzugeben. Im Grunde handelt es sich um den wieder eingeführten – und leicht überarbeiteten – *Normalmodus* der Flash-Versionen 5 und MX, der in Flash MX 2004 vorübergehend abgeschafft worden war.

Das Hinzufügen von ActionScript-Befehlen und -Funktionen erfolgt – wie auch im Standardbetrieb möglich – aus dem Fensterbereich links oben oder mit Hilfe der Plus-Schaltfläche in der Symbolleiste, die bei eingeschalteter Skripthilfe übrigens weniger Elemente enthält. Beachten Sie, dass manche Einträge in den Menüs mehrere reale ActionScript-Funktionen bündeln. Beispielsweise fasst die hier beschriebene Auswahl *goto* sämtliche Einzelfunktionen zusammen, die weiter unten im Unterabschnitt *Sprungbefehle* behandelt werden. Die manuelle Code-Eingabe ist in diesem Modus übrigens gar nicht möglich.

Es würde zu weit führen, hier die Eingabe vieler verschiedener Elemente über die Skripthilfe zu beschreiben. Da der Rest des Kapitels die Syntax zahlreicher ActionScript-Anweisungen beschreibt, ist dies auch nicht nötig – wer ActionScript beherrscht, kommt notfalls auch mit der Skripthilfe zurecht. Deshalb wird hier als Beispiel nur die Auswahl der Anweisung *goto* beschrieben.

In Abbildung 6-5 sehen Sie die verfügbaren Parameter für die Aktion *goto*. Wählen Sie *Globale Funktionen* → *Zeitleistensteuerung* → *goto*, um sie an der aktuellen Skriptposition hinzuzufügen. Im Parameterbereich können Sie folgende Einstellungen vornehmen:

- Zunächst können Sie sich für eine der beiden Optionen *Gehe zu und abspielen* oder *Gehe zu und stoppe* entscheiden – der Film wird am Sprungziel weiter abgespielt beziehungsweise angehalten.
- Darunter können Sie eine *Szene* auswählen. Szenen bieten eine praktische Möglichkeit, Flash-Filme zu strukturieren; sie werden weiter unten besprochen.
- Der *Typ* bestimmt, wie das Zielbild angegeben wird: *Bildnummer* ist die in der Zeitleiste ablesbare Nummer des gewünschten Bildes. Mit *Bildbezeichnung* springen Sie zu einem Schlüsselbild, dem in der Eigenschaftenleiste ein Name zugeordnet wurde. *Ausdruck* ermöglicht die Angabe eines beliebigen Ausdrucks, der durch Berechnung oder Zusammenfügung eine Bildnummer bezie-

hungsweise einen Bildnamen ergibt. Zu guter Letzt können Sie noch *Nächstes Bild* oder *Voriges Bild* wählen – relative Sprünge ohne jegliche Parameter.

- Als Letztes wird *Bild* angegeben, zu dem der Sprung erfolgen soll; je nach gewähltem Typ in unterschiedlichem Format. Für den Typ Bildbezeichnung können Sie einen der verfügbaren Bezeichner aus dem Pull-down-Menü auswählen.

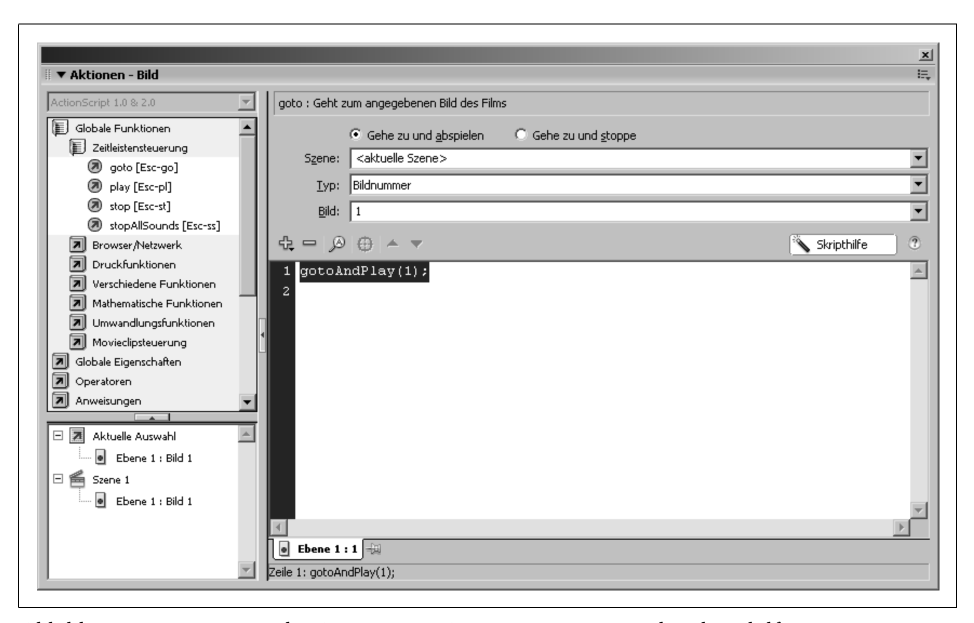


Abbildung 6-5: Parameter der ActionScript-Anweisung »goto« in der Skripthilfe



Wenn Ihr manuell eingegebener ActionScript-Code Syntaxfehler enthält, verweigert Flash den Wechsel in die Skripthilfe, bis Sie die Fehler beseitigt haben. Zeilennummern und nähere Informationen über die Fehler erhalten Sie, wenn Sie *Syntax überprüfen* wählen.

ActionScript-Grundelemente

In diesem Abschnitt erhalten Sie eine Einführung in wichtige ActionScript-Grundelemente. Hier werden unter anderem die wichtigen Anweisungen zur Filmsteuerung vorgestellt, anschließend werden die klassischen Elemente einer Programmiersprache wie Variablen (benannte Zwischenspeicher), Fallentscheidungen (*wenn ..., dann ...*) und Schleifen (*mache ... n-mal*, *mache ... solange ...*) eingeführt.



Falls Sie noch keine oder sehr wenig Programmiererfahrung haben, kommen Ihnen die Beschreibungen hier möglicherweise ein wenig theoretisch vor. In diesem Fall können Sie auch zuerst die Praxisbeispiele am Ende dieses Kapitels durcharbeiten und dann hierher zurückkehren, um die Theorie dahinter zu verstehen.

Die ActionScript-Syntax

Jede Programmiersprache besitzt Regeln dafür, wie Anweisungen geschrieben werden müssen. Damit sie ohne übertriebenen Aufwand erfolgreich von einem Computer verarbeitet werden kann, sind diese Regeln üblicherweise sehr streng und bieten nur wenige Alternativen, ganz im Gegensatz zu der Variationsvielfalt und der hohen Fehlertoleranz, die die natürlichen Sprachen der Menschen erlauben.

Die grundlegende Syntax von ActionScript stimmt durch die ECMA-Zertifizierung mit derjenigen von JavaScript überein. Dies hat wiederum zur Folge, dass viele Ideen dieser Syntax aus der Tradition der verbreiteten Programmiersprache C stammen. Wenn Sie sich die folgenden einfachen Syntaxregeln merken, sollten Sie mit ActionScript ohne Probleme zurechtkommen.

Anweisungen

Ein einzelner Arbeitsschritt in einem Skript wird als *Anweisung* bezeichnet. Jede Anweisung steht in einer eigenen Zeile und endet mit einem Semikolon. Beispiele:

```
gotoAndPlay (25);
myMovieClip.stop();
```

Anweisungen enthalten oftmals Funktionsaufrufe. Dabei kann es sich sowohl um eingebaute als auch um selbst geschriebene Funktionen handeln – Letztere werden weiter unten behandelt. In beiden Fällen wird die Funktion einfach durch ihren Namen aufgerufen; darauf folgen stets runde Klammern für die *Argumente* (d.h. die näheren Bestimmungen) der Funktion. Das gilt auch dann, wenn eine Funktion keine Argumente entgegennimmt; in diesem Fall bleiben die Klammern einfach leer:

```
gotoAndStop (17);
stop();
```

Blöcke

Bei einer Fallentscheidung hängt normalerweise genau eine Anweisung von der Bedingung ab; auch bei Schleifen wird nur eine einzelne Anweisung wiederholt. In vielen Fällen soll aber eine längere Abfolge von Anweisungen in einem solchen Kontext stehen. Zu diesem Zweck werden Anweisungsblöcke verwendet: Wenn Sie eine Folge von Anweisungen in geschweifte Klammern { } setzen, verhalten sich diese nach außen wie eine einzelne Anweisung. Jede Anweisung innerhalb des Blocks

wird wieder mit einem Semikolon abgeschlossen; der Block selbst dagegen nicht. Im folgenden Beispiel wird der Sound nur dann angehalten, wenn die Variable `punkte` einen größeren Wert als 100 hat; der Sprung zum Bild `"weiter"` erfolgt dagegen immer:

```
// Sound anhalten, wenn punkte > 100
if (punkte > 100)
   stopAllSounds();
// Immer Sprung zum Frame "weiter"
gotoAndPlay ("weiter");
```

Soll dagegen auch der Sprung nur stattfinden, wenn die Bedingung zutrifft, dann müssen Sie einen Block benutzen:

```
// Sound anhalten und Sprung, wenn punkte > 100
if (punkte > 100) {
   stopAllSounds();
   gotoAndPlay ("weiter");
}
```

Die geschweiften Klammern erzeugen Sie auf einem Windows-Rechner mit **Alt Gr + 7** beziehungsweise **Alt Gr + 0**; auf einem Mac dagegen über die Tastenkombinationen **ALT + 5** und **ALT + 6**.

Neben der gezeigten Schreibweise, bei der die öffnende geschweifte Klammer in derselben Zeile steht wie ihr »Auslöser« (hier das `if`), gibt es auch die Möglichkeit, sie allein in die nächste Zeile zu setzen:

```
// Sound anhalten und Sprung, wenn punkte > 100
if (punkte > 100)
{
   stopAllSounds();
   gotoAndPlay ("weiter");
}
```

In beiden Fällen wird der Inhalt der geschweiften Klammern normalerweise eingerückt, um zu kennzeichnen, dass dieser Programmcode gewissermaßen auf einer »spezielleren Ebene« stattfindet. Es ist Geschmackssache, für welche der beiden Varianten Sie sich entscheiden. In diesem Buch wird bei Fallentscheidungen, Schleifen und Ereignisprozeduren – `on()` und `onClipEvent()` – dieselbe Zeile, bei Funktionen- und Klassendefinitionen dagegen die nächste Zeile verwendet. Dies ist eine weit verbreitete Lösung, und in den weiter oben erwähnten Autoformat-Optionen finden Sie entsprechende Einstellungsmöglichkeiten.



In der Praxis empfiehlt es sich bei Fallentscheidungen oder Schleifen, selbst eine einzelne Anweisung von vornherein in einen Block zu setzen. Dies verbessert die Übersichtlichkeit und verhindert Flüchtigkeitsfehler, wenn Sie später noch eine weitere Anweisung für diesen Kontext hinzufügen möchten.

Bezeichner

Wenn Sie selbst Variablen oder Funktionen definieren, benötigen diese einen eindeutigen Namen, den *Bezeichner*. Er muss mit einem Buchstaben (a–z, A–Z; keine Umlaute) oder _ (Unterstrich) beginnen; darauf können beliebig viele dieser Zeichen sowie Ziffern von 0 bis 9 folgen. Gültige Bezeichner sind also beispielsweise `test3`, `_abc` oder `a_4`. Achten Sie bei der Wahl von Bezeichnern auf aussagekräftige Namen – beispielsweise sollte eine Punktzahl eher `punkte` heißen und nicht `p`.

Variablen- und Funktionsbezeichner sollten mit einem Kleinbuchstaben beginnen; mit Großbuchstaben fangen üblicherweise die Namen von Klassen an, die Sie im nächsten Kapitel kennen lernen. Wenn Ihre Bezeichner aus mehreren Wörtern bestehen, sollte jedes Anschlusswort mit einem Großbuchstaben beginnen. Beispiele: `meinePunkte`, `autoKmStand`. Dies macht die Namen lesbarer und hilft Ihnen, sich die Schreibweise zu merken. Eine ältere Konvention benutzt nur Kleinbuchstaben und trennt die einzelnen Wörter durch Unterstriche: `meine_punkte`, `auto_km_stand`. Es ist egal, welche der beiden Varianten Sie wählen – Sie sollten sich nur konsequent für eine von ihnen entscheiden.



Selbstverständlich dürfen Ihre eigenen Bezeichner nicht mit den Namen der bestehenden ActionScript-Schlüsselwörter, -Funktionen und -Objekte kollidieren. Glücklicherweise werden Sie aber vom ActionScript-Compiler gewarnt, falls Ihnen dies passieren sollte.

Kommentare

In jeder Programmiersprache gibt es eine Möglichkeit, Kommentare zu setzen. Dies sind Textzeilen oder -blöcke, die vom Rechner ignoriert werden. Guter Programmcode sollte immer mit aussagekräftigen Kommentaren versehen werden, damit Sie sich auch bei späteren Änderungen noch zurechtfinden oder den Code zur Weiterbearbeitung an andere Programmierer weitergeben können. ActionScript unterstützt zwei verschiedene Kommentarsorten:

- *Einzeiliger Kommentar:* Hinter zwei Schrägstrichen (`//`) wird der restliche Text einer Zeile ignoriert. Beispiele:

  ```
  // Dies ist ein Kommentar
  stop();  // Kommentar nach Anweisung
  ```

  Beachten Sie, dass die Autoformat-Funktion Kommentare grundsätzlich in eigene Zeilen verschiebt.
- *Mehrzeiliger Kommentar:* Sie können Inhalte zwischen `/*` und `*/` setzen, um sie über beliebig viele Zeilen als Kommentar zu betrachten. Dies ist für längere Beschreibungen von Funktionen nützlich, aber auch, um vorübergehend einen Codeabschnitt zu deaktivieren und eine Alternative zu testen. Hier ein Beispiel:

  ```
  /* Wenn die Variable runden mindestens 8 ist,
     soll der Sprung zum Bild "ende" erfolgen */
  ```

```
if (runden >= 8) {
   gotoAndPlay ("ende");
}
```

Schaltflächen

Eines der wichtigsten interaktiven Elemente für Flash-Filme ist das *Schaltflächen-Symbol*. Sie können definieren, was passieren soll, wenn eine Schaltfläche angeklickt (oder anderweitig aktiviert) wird. Auf diese Weise wird aus einem linearen Film eine interaktive Präsentation oder Multimedia-Anwendung.

Wenn Sie eine neue Schaltfläche erstellen möchten, funktioniert dies über den Menübefehl *Einfügen → Neues Symbol* beziehungsweise mit Hilfe der Tastenkombination **STRG + F8**. Wählen Sie im bekannten Dialog *Neues Symbol erstellen* unter *Verhalten* die Option *Schaltfläche*.

Die Zeitleiste eines Schaltflächen-Symbols sieht ein wenig anders aus als diejenige des Hauptfilms oder eines Movieclips: Statt der nummerierten Bilder sind vier breite, benannte Bilder zu sehen. Sie tragen die Bezeichnungen *Auf*, *Darüber*, *Gedrückt* und *Aktiv* (siehe Abbildung 6-6).

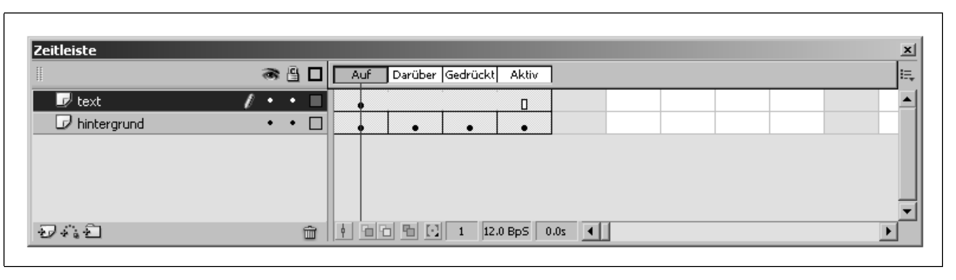


Abbildung 6-6: Die spezielle Zeitleiste eines Schaltflächen-Symbols

Diese Bilder repräsentieren die unterschiedlichen Zustände des Buttons:

- *Auf* wird angezeigt, solange die Schaltfläche nicht von der Maus berührt wird (Grundzustand).
- *Darüber:* Sobald der Mauszeiger den Button berührt, wechselt er zu diesem Bild.
- *Gedrückt:* Dieser Zustand wird aktiv, sobald die Maustaste auf dem Button heruntergedrückt wird.
- *Aktiv:* Dies ist *kein* separater Zustand, sondern die Definition des sensitiven Bereichs der Schaltfläche. Wenn die Bilder der drei Zustände unterschiedliche Größen besitzen oder nicht durchgehend gefüllt sind, sollten Sie hier ein Schlüsselbild einfügen und eine durchgehende Füllung erstellen, die sämtliche Bereiche der anderen Zustände bedeckt. Dabei kann Ihnen der Zwiebelschaleneffekt helfen.

Übrigens können Sie einer Schaltfläche auch ganz leicht einen Klick-Sound zuweisen: Aktivieren Sie das Schlüsselbild im Bild *Gedrückt* (bzw. erstellen Sie zunächst eines). Wählen Sie nun in der Eigenschaftenleiste den passenden Sound aus, und stellen Sie die Option *Sync* auf *Ereignis*.

Soll einer der Zustände animiert sein, so funktioniert dies nicht direkt. Ziehen Sie stattdessen einen Movieclip mit der gewünschten Animation in das passende Bild des Schaltflächen-Symbols.

Wenn Sie eine Schaltfläche auf die Bühne ziehen, können Sie die neue Instanz anklicken und ihr in der Aktionen-Palette ein Skript nach folgendem Schema zuweisen:

```
on (Button-Ereignis) {
   Anweisung(en);
}
```

Als `Button-Ereignis` wird am häufigsten `release` eingesetzt. Es tritt ein, nachdem die Maustaste auf dem Button gedrückt und wieder losgelassen wurde. Dies entspricht dem Verhalten, das die meisten Benutzer von einem Button erwarten. Das folgende Beispiel springt zum Bild Nummer 50, sobald die Maustaste auf dem Button losgelassen wird:

```
on (release) {
   gotoAndPlay (50);
}
```

Es ist übrigens kein Problem, auf mehrere Ereignisse mit denselben Anweisungen zu reagieren. Das folgende Beispiel reagiert sowohl auf das Loslassen der Maustaste als auch auf das Drücken der Leertaste:

```
on (release, keyPress "<Space>") {
   gotoAndPlay ("weiter");
}
```

Tabelle 6-2 zeigt sämtliche möglichen Schaltflächen-Ereignisse im Überblick.

Tabelle 6-2: Ereignisse für Schaltflächen-Aktionen

Ereignis	Erläuterung
on (press)	Maustaste drücken
on (release)	Maustaste drücken und wieder loslassen
on (releaseOutside)	Maustaste drücken; außerhalb der Schaltfläche loslassen
on (rollOver)	Mauszeiger berührt die Schaltfläche
on (rollOut)	Mauszeiger verlässt die Schaltfläche
on (dragOver)	Mauszeiger berührt die Schaltfläche bei gedrückter Maustaste
on (dragOut)	Mauszeiger verlässt die Schaltfläche bei gedrückter Maustaste
on (keyPress "<Left>")	**Pfeiltaste links** gedrückt
on (keyPress "<Right>")	**Pfeiltaste rechts** gedrückt
on (keyPress "<Home>")	**Pos1** gedrückt

Tabelle 6-2: Ereignisse für Schaltflächen-Aktionen (Fortsetzung)

Ereignis	Erläuterung
on (keyPress "<End>")	**Ende** gedrückt
on (keyPress "<Insert>")	**Einfg** gedrückt
on (keyPress "<Delete>")	**Entf** gedrückt
on (keyPress "<Backspace>")	**Rücktaste** gedrückt
on (keyPress "<Enter>")	**Enter** gedrückt
on (keyPress "<Up>")	**Pfeiltaste oben** gedrückt
on (keyPress "<Down>")	**Pfeiltaste unten** gedrückt
on (keyPress "<PageUp>")	**Bild auf** gedrückt
on (keyPress "<PageDown>")	**Bild ab** gedrückt
on (keyPress "<Tab>")	**Tab** gedrückt
on (keyPress "<Escape>")	**Escape**-Taste gedrückt
on (keyPress "<Space>")	**Leertaste** gedrückt
on (keyPress "*Taste*")	angegebene alphanumerische Taste gedrückt



Das Ereignis `keyPress` ist besonders wichtig, wenn Sie FlashLite-Anwendungen für Smartphones exportieren – da sie nicht mit einer Maus ausgestattet sind, können Steuerungsaufgaben nur per Tastendruck durchgeführt werden.

Zu der direkten Zuweisung von Ereignissen an Button-Instanzen gibt es eine interessante Alternative, die Ihnen hilft, Ihren Code an weniger Stellen zu sammeln: Wenn Sie einer Schaltflächen-Instanz einen Instanznamen zuweisen, können Sie ihr in einem Bildskript eine *Ereignisprozedur* (englisch Event-Handler) zuordnen. Klicken Sie die Instanz dazu an, und geben Sie links oben in der Eigenschaftenleiste – unter dem Eintrag *Schaltfläche* – einen eindeutigen Namen ein (siehe Abbildung 6-7). Er sollte den weiter oben angesprochenen Regeln für ActionScript-Bezeichner genügen. Momentan soll der Instanzname `weiter` als Beispiel dienen.

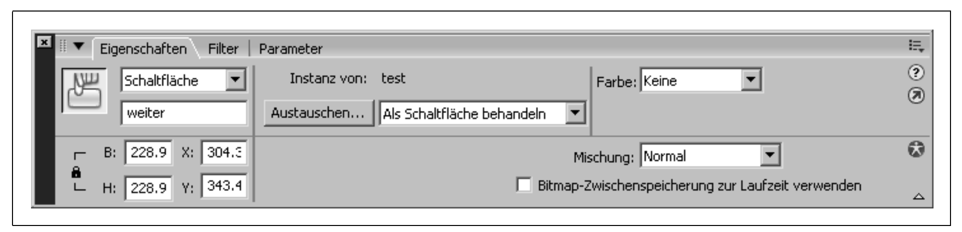


Abbildung 6-7: Zuweisung eines Instanznamens für eine Schaltfläche

Nun können Sie in einem Bild-Skript der Zeitleiste, in der sich die Instanz befindet, die Ereignisprozedur erstellen. Ihr Name ist `*Instanzname*.on*Ereignis*`, wobei *`Ereig-`*

`nis` jeweils das großgeschriebene Ereignis aus Tabelle 6-2 ist. Wenn die Schaltflächen-Instanz `weiter` auf das Ereignis `release` reagieren soll, sieht die passende Prozedur-Zuweisung folgendermaßen aus:

```
weiter.onRelease = function() {
   gotoAndPlay (100);
}
```

Dem Ereignis wird also eine anonyme Funktion zugewiesen. Alternativ können Sie auch eine benannte Funktion verwenden – das ist vor allem dann nützlich, wenn Sie ein und dieselbe Funktion mehreren Ereignissen zuweisen möchten. Die Definition von Funktionen wird weiter unten besprochen; hier nur das nicht weiter erläuterte Beispiel:

```
// Der Schaltfläche weiter die Funktion sprung() zuordnen
weiter.onRelease = sprung;

// Definition der Funktion sprung()
function sprung ()
{
   gotoAndPlay (100);
}
```

Zeitleistensteuerung

Die wichtigsten und einfachsten ActionScript-Anweisungen, die in diesem Buch schon mehrfach intuitiv verwendet wurden, steuern den Ablauf der Bilder einer Zeitleiste. Standardmäßig werden die Zeitleisten des Hauptfilms und eventueller Movieclips linear abgespielt. Für die Navigation in einer interaktiven Präsentation ist es wichtig, dass dieser Ablauf unterbrochen werden kann. Die in diesem Unterabschnitt vorgestellten Funktionen können sowohl im Hauptfilm als auch innerhalb eines Movieclips eingesetzt werden. Sie steuern jeweils die Zeitleiste, in der sie sich gerade befinden. Der Zugriff auf andere Zeitleisten – etwa von verschachtelten Movieclips – wird im nächsten Abschnitt erläutert.

Anhalten und Weiterspielen

Die einfachsten Steueranweisungen sind `play()` und `stop()`. Wenn Sie den Film an einer bestimmten Stelle anhalten möchten, können Sie die folgende Anweisung benutzen:

```
stop();
```

Wenn diese Anweisung im Skript eines Schlüsselbildes steht, hält der Film automatisch an der entsprechenden Stelle an. Auf diese Weise erstellen Sie also ein Standbild. Wann immer Sie ein Menü, eine Info-Seite oder ähnliche statische Bildschirme erstellen möchten, ist dies die passende Lösung. Um diesen Bildschirm wieder zu verlassen und den Film weiter abzuspielen, wird die Anweisung `play()` benutzt; Sie können sie beispielsweise einer Schaltfläche zuweisen.

Hier eine einfache Möglichkeit, `play()` und `stop()` auszuprobieren:

1. Erstellen Sie in einem neuen Film eine beliebige Animation.
2. Legen Sie zwei Schaltflächen mit den Beschriftungen *Stop* bzw. *Weiter* an.
3. Richten Sie eine neue Ebene namens *buttons* ein.
4. Ziehen Sie die Schaltfläche *Stop* auf die Bühne, und weisen Sie ihr folgendes Skript zu:

   ```
   on (release) {
      stop();
   }
   ```

5. Ziehen Sie auch den Button *Weiter* auf die Bühne; er erhält die folgenden Anweisungen:

   ```
   on (release) {
      play();
   }
   ```

Wenn Sie diesen Film mittels *Film testen* ausprobieren, werden Sie feststellen, dass Sie die Animation jederzeit mit *Stop* unterbrechen und mit *Weiter* fortsetzen können.

Sprungbefehle

Mit `play()` und `stop()` lassen sich zwar schon durchblätterbare Standbilder erstellen, aber der Film bleibt nach wie vor linear. Eine interaktive Auswahl von Inhalten ist erst möglich, wenn Sie zu verschiedenen Bildern eines Films springen können. Zu diesem Zweck gibt es vor allem zwei wichtige Funktionen:

- `gotoAndPlay()` springt zum angegebenen Frame der aktuellen Zeitleiste und spielt den Film von dort weiter ab.
- `gotoAndStop()` springt ebenfalls zum gewünschten Bild, hält den Film aber dort an.

Im einfachsten Fall ist das Argument – die Angabe des Bildes, zu dem Sie springen möchten – eine einfache Nummer, wie sie in der Zeitleiste steht. Die folgende Anweisung springt zum Bild Nummer 100 der aktuellen Zeitleiste und spielt diese dann ab:

```
gotoAndPlay (100);
```

Das folgende Beispiel springt dagegen zu Frame 50 und bleibt dort stehen:

```
gotoAndStop (50);
```

Praktischer als die Verwendung von Bildnummern sind übrigens *Bildmarkierungen*. Erstens kann man sich eine aussagekräftige Bezeichnung wie `"menue"` leichter merken als beispielsweise 356, und außerdem ändern sich Markierungen auch dann nicht, wenn Sie später Bilder einfügen oder entfernen.

Wenn Sie eine Bildmarkierung anbringen möchten, müssen Sie ein Schlüsselbild anklicken und können dann links oben in der Eigenschaftenleiste einen Namen eingeben. Das entsprechende Bild wird in der Zeitleiste durch ein kleines Fähnchen mit der Bezeichnung gekennzeichnet. Unter *Beschriftungstyp* können Sie die Art der jeweiligen Markierung einstellen:

- *Name* erstellt eine Bildbezeichnung, wie sie für die hier verwendeten Sprungbefehle benötigt wird. Das Symbol in der Zeitleiste ist ein rotes Fähnchen.
- *Kommentar* dient lediglich Ihrer eigenen Orientierung in der Zeitleiste; als Symbol dienen grüne Kommentarzeichen (//).
- *Anker* können als HTML-Sprungziele exportiert werden; ihr Symbol ist ein orangefarbener Anker.

Genau wie für Bildaktionen oder Sounds sollten Sie auch für Bildmarkierungen eine eigene Ebene verwenden. In Abbildung 6-8 sehen Sie Bildmarkierungen in der Zeitleiste und in der Eigenschaftenleiste.

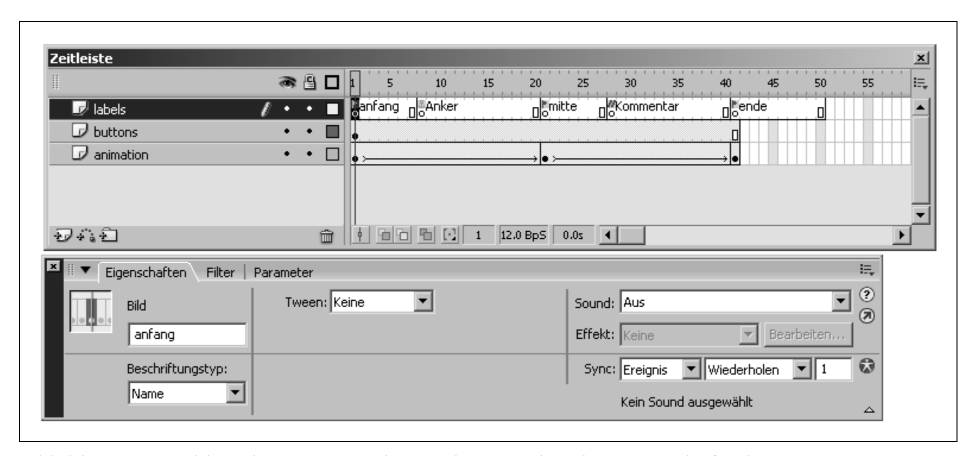


Abbildung 6-8: Bildmarkierungen in der Zeitleiste und in der Eigenschaftenleiste

Wenn Sie einen Sprung zu einem Bild mit einer Markierung durchführen möchten, müssen Sie die Bezeichnung in Anführungszeichen setzen. Das folgende Beispiel springt zu dem Frame mit der Markierung `"menue"`:

```
gotoAndPlay ("menue");
```

Wenn Sie Ihren Film übersichtlicher gestalten möchten, können Sie *Szenen* verwenden. In jeder Szene wird eine neue, aufgeräumte Zeitleiste angezeigt, die Sie wie gewohnt mit Ebenen und Schlüsselbildern ausstatten können. Zur Verwaltung von Szenen wird das Bedienfeld *Szene* benutzt; Sie können es mit dem Befehl *Fenster → Andere Bedienfelder → Szene* oder der Tastenkombination **Umschalt + F2** öffnen. In Abbildung 6-9 wird die Palette gezeigt.



Abbildung 6-9: Das Bedienfeld »Szene«

Die drei Schaltflächen in der unteren Leiste haben von links nach rechts folgende Bedeutung:

- *Szene duplizieren*. Eine Kopie der aktuellen Szene wird erstellt. Das ist beispielsweise nützlich, wenn Sie eine längere Sequenz nachträglich in Szenen aufteilen möchten: Sie können sie mehrfach duplizieren und dann jeweils unterschiedliche Teile der Zeitleiste löschen.
- *Neue Szene:* Eine neue, leere Szene wird eingefügt. Dies funktioniert alternativ auch über den Menübefehl *Einfügen* → *Szene*.
- *Szene löschen:* Die aktuelle Szene wird (mit Sicherheitsnachfrage) gelöscht.

Um zu einer bestimmten Szene zu wechseln, brauchen Sie diese lediglich anzuklicken. Sie können sie aber auch aus dem Szenen-Pop-up-Menü am Kopf des Dokumentfensters auswählen. Per Doppelklick in der Szenen-Palette lässt sich eine Szene umbenennen; die Reihenfolge können Sie durch einfaches Ziehen nach oben oder unten verändern. Im fertigen Film werden standardmäßig alle Szenen nacheinander abgespielt.

Für die Sprungbefehle haben Szenen natürlich ebenfalls eine Bedeutung: Sie können bei `gotoAndStop()` und `gotoAndPlay()` auf Wunsch die beiden Argumente *Szene* und *Bild* angeben. Wenn Sie zu Bild 1 in der Szene `"hauptfilm"` springen möchten, sieht dies beispielsweise so aus:

```
gotoAndPlay ("hauptfilm", 1);
```

Ein Sprung zum Frame `"menue"` in der Szene `"intro"` funktioniert dagegen so:

```
gotoAndStop ("menue", "intro");
```

Übrigens sind Bildmarkierungen auch über Szenengrenzen hinweg gültig. Sie können diese Anweisung also auch einfach folgendermaßen schreiben:

```
gotoAndStop ("intro");
```



Die Bilder eines Films werden auch dann durchnummeriert, wenn Sie Szenen verwenden. Alternativ zur Szenen-Syntax können Sie also auch immer die eigentliche Bildnummer angeben. Angenommen, Ihr Film besitzt zwei Szenen namens *intro* und *hauptfilm*. *intro* besteht aus 20 Bildern (es zählt jeweils die Ebene mit den meisten Bildern). Die beiden folgenden Befehle bewirken jeweils einen Sprung zu Bild 1 von *hauptfilm*:

```
gotoAndPlay ("hauptfilm", 1);   // mit Szenenangabe
gotoAndPlay (21);               // reine Bildnummer
```

Es gibt noch einige weniger häufig verwendete Sprungbefehle, die sich vornehmlich zum einfachen »Blättern« eignen. Der Vollständigkeit halber sollen sie hier erwähnt werden:

```
nextFrame();     // ein Bild weiter
prevFrame();     // ein Bild zurück
nextScene();     // eine Szene weiter
prevScene();     // eine Szene zurück
```

Im Zusammenhang mit der Zeitleistensteuerung steht noch eine andere Funktion zur Verfügung; Sie finden sie auch im ActionScript-Auswahlmenü unter *Globale Funktionen → Zeitleistensteuerung*:

```
stopAllSounds();
```

Diese einfache Anweisung beendet sofort das Abspielen sämtlicher Sounds. Sie ist im Zusammenhang mit der Filmsteuerung sinnvoll, weil unterschiedliche Szenen, Abschnitte oder Standbilder manchmal mit verschiedener Hintergrundmusik oder gesprochenen Beschreibungen ausgestattet sind.

Variablen, Ausdrücke und Operationen

Programmierung wäre undenkbar, wenn Programmiersprachen nicht in der Lage wären, komplexe Ausdrücke zu berechnen. Im Grunde macht diese Fähigkeit das Schreiben von Programmen zur angewandten Mathematik: Auch hier müssen Sie sich mit der Formulierung von Rechenausdrücken beschäftigen. Allerdings haben Sie es leichter als ein Mathematiker, weil Sie die Ausdrücke in beliebig komplexer Form hinschreiben und ihre Vereinfachung dem Computer überlassen können.



Allgemein gilt: Überall, wo in einer Programmiersprache die Angabe eines konkreten Wertes erwartet wird, kann stattdessen auch ein beliebig komplexer Ausdruck stehen, solange er einen Wert des geforderten Typs ergibt.

In diesem Unterabschnitt lernen Sie Ausdrücke und ihre Bestandteile kennen:

- *Literale:* Dies sind »wörtlich« gemeinte Komponenten wie Zahlen oder Strings (Textblöcke).
- *Variablen:* Eine Variable ist ein benannter Speicherplatz. Wenn Sie sie in einen Ausdruck einsetzen, wird der aktuell in ihr gespeicherte Wert zur Berechnung verwendet.
- *Operatoren:* Hier handelt es sich um Verknüpfungsvorschriften, mit denen Literale und Variablen zu komplexeren Ausdrücken verbunden werden.
- *Funktionsaufrufe:* Da auch viele Funktionen ein Ergebnis liefern, können Sie sogar diese in Ausdrücken unterbringen.



ActionScript stellt eine praktische Funktion zum Testen von Ausdrücken zur Verfügung:

```
trace (Ausdruck);
```

Die Ausgabe des ausgewerteten Ausdrucks erfolgt in das so genannte *Ausgabefenster* (*Fenster* → *Ausgabe* oder **F2**). `trace()` ist äußerst praktisch für die Fehlersuche in Skripten.

Damit Sie vor der endgültigen Veröffentlichung nicht mühevoll alle `trace()`-Anweisungen aus Ihren Skripten entfernen müssen, können Sie beim Exportieren die Option *Trace-Aktionen übergehen* aktivieren (siehe Kapitel 5).

Literale

Wenn Funktionen Argumente benötigen, erwarten sie zunächst einfache, nicht weiter zu berechnende Werte. Beispielsweise wird ein Sprung zu Frame Nummer 3 mit folgender Anweisung durchgeführt:

```
gotoAndPlay (3);
```

Die `3` ist hier ein numerisches Literal, also eine Zahl, die nicht mehr weiter ausgewertet werden muss.

Allein stehende Literale sind die einfachsten möglichen Ausdrücke. ActionScript kennt folgende Arten von Literalen:

- *Numerische Literale* – Zahlen
- *String-Literale* – Zeichenketten (Textblöcke)
- *Boolesche Literale* – `true` und `false` für Wahrheitswerte
- `undefined` – der Wert einer undefinierten Variablen
- `null` – ein leeres Objekt

Für die Angabe von Frames, Objektpositionen und -größen sowie für Berechnungen aller Art werden Zahlen benötigt. Je nach Bedarf kann es sich dabei um ganze

Zahlen (englisch Integer) oder Fließkommazahlen handeln. Integer-Literale können in drei verschiedenen Zahlenformaten angegeben werden:

- Einfache Zahlen wie `7`, `2048` oder `-678` werden *dezimal* interpretiert.
- Durch eine vorangestellte `0` wird die Zahl *oktal*, gehört also zum Achtersystem. Hier gibt es nur die Ziffern 0 bis 7; der Wert 8 wird als 10 dargestellt und so weiter. Beispielsweise entspricht `077` dem dezimalen Wert 63, `1000` bedeutet 512, und `-432` steht für -282.
- Wenn Sie `0x` vor eine Zahl setzen, wird sie dem *Hexadezimalsystem* (Sechzehnersystem) zugerechnet. Hier gibt es sechzehn verschiedene Ziffern: 0–9 wie gehabt; zusätzlich stehen die Buchstaben A–F (oder auch a–f) für 10 bis 15. Die Stellen versechzehnfachen ihren Wert nach links hin. `0xF` ist also 15, `0x10` steht für 16, `0xF0` bedeutet 240, und `-0xABC` ist -2748.

Dezimalbrüche werden in Programmiersprachen als *Fließkommazahlen* (englisch Floating Point Numbers) bezeichnet. Da der Computer jede Zahl in einem Speicherbereich mit einer bestimmten Maximalgröße ablegen muss, kennt er nur abbrechende Dezimalbrüche mit begrenzter Genauigkeit; periodische oder nicht-abbrechende Dezimalzahlen widersprechen seinen technischen Gegebenheiten. Beachten Sie, dass das Dezimaltrennzeichen nicht das Komma, sondern gemäß amerikanischer Schreibweise der Punkt ist. Beispiele für Fließkomma-Literale sind `3.8762` oder `-0.1429`.

Zur Textausgabe oder auch zur Überprüfung von Benutzereingaben verwendet ActionScript Text in Form von Zeichenketten (englisch *Strings*). Auch Frames können beispielsweise in Form von Strings angegeben werden, wenn Sie ihnen in der Zeitleiste eine benannte Bildmarkierung zuweisen.

Ein String-Literal steht grundsätzlich in Anführungszeichen – im Gegensatz zu vielen anderen Programmiersprachen ist es bei ActionScript egal, ob Sie `'einfache'` oder `"doppelte"` Anführungszeichen benutzen. Die direkte Abfolge zweier Anführungszeichen (`""`) ist der leere String; er besteht aus 0 Zeichen. Hier einige weitere Beispiele für Strings: `"Ein Text"`, `'300'`, `"!§$%&"`.

Neben gewöhnlichen Zeichen kann ein String noch so genannte *Escape-Sequenzen* enthalten, Zeichenfolgen mit besonderer Bedeutung, die mit einem Backslash (\) beginnen: `\"` und `\'` ermöglichen die Verwendung der Anführungszeichen selbst, `\\` ist der Backslash als Zeichen. `\n` stellt einen Zeilenumbruch dar, `\t` einen Tabulator. Außerdem können Sie ein Zeichen durch seinen numerischen ASCII-Code angeben: `\00` bis `\377` interpretiert den Wert oktal (im Achtersystem) und gibt so die Zeichen 0 bis 255 an; `\x00` bis `\xFF` sind die entsprechenden Hexadezimalcodes.

Die Booleschen Literale `true` und `false` sind das überprüfbare Ergebnis logischer Operationen: Der Vergleich `3 < 5` hat den Wert `true`, während `3 > 5` eindeutig `false` ist. Wenn Sie möchten, können Sie diese Werte aber auch explizit benutzen und beispielsweise in Variablen speichern.

Variablen

In der Vorzeit der Computertechnik mussten sich Programmierer selbst mit der Ablage ihrer Nutzdaten im Arbeitsspeicher abmühen. Die dazu eingesetzten (numerischen) Speicheradressen gibt es mit gewissen technischen Verbesserungen (zum Beispiel virtuelle Adressierung mit Auslagerung auf die Festplatte) noch heute. Allerdings findet diese Form der Speicherzuteilung fast immer hinter den Kulissen statt. Dies ist nicht zuletzt das Verdienst der Variablen, die in beinahe jeder Programmiersprache zur Verfügung stehen. Dies nur vorweg – falls Sie finden, dass Variablen schwierig zu verstehen sind, denken Sie daran, dass sie eine enorme Arbeitserleichterung darstellen: Sie brauchen sich nicht mehr zu merken, *wo* Sie etwas gespeichert haben, sondern nur noch, *unter welchem Namen* es abgelegt wurde.

Eine Variable ist ein Bereich im Arbeitsspeicher, den Sie durch einen selbst gewählten Bezeichner kennzeichnen können. Durch den Operator `=` wird einer Variablen ein Wert zugewiesen. Im folgenden Beispiel erhält eine Variable namens `a` den Wert 5:

```
a = 5;
```

Es ist empfehlenswerter, eine Variable bei der ersten Verwendung mit dem Schlüsselwort `var` zu *deklarieren*, das heißt, ihre künftige Verwendung anzukündigen. Dies kann entweder separat geschehen oder zusammen mit einer Wertzuweisung. Die explizite Deklaration sieht so aus:

```
// Variable b deklarieren
var b;
// b einen Anfangswert zuweisen
b = 7;
```

Eine integrierte Deklaration mit Wertzuweisung hat dagegen folgendes Format:

```
var c = "Hallo";
```

Durch die Deklaration ordnen Sie eine Variable ausdrücklich einem *Geltungsbereich* zu: Jede Variable gilt ausschließlich innerhalb derjenigen Zeitleiste, Funktion oder Klasse, in der sie deklariert wurde. Wie Sie auf die Variablen anderer Zeitleisten zugreifen können, erfahren Sie im nächsten Abschnitt.

Standardmäßig besitzt eine Variable in ActionScript keinen festgelegten Datentyp. Sie können beispielsweise ohne Weiteres zuerst eine Zahl und später einen String darin speichern:

```
// Deklarieren und Zahl speichern
var d = 256;
// String speichern
d = "Guten Morgen";
```

In ActionScript 2.0, der in Flash MX 2004 neu eingeführten Version, können Sie Variablen aber auf Wunsch auch *typisiert* benutzen: Bei der Deklaration lässt sich von vornherein ein *Datentyp* festlegen, dem die Variable dauerhaft entsprechen

muss. Der Datentyp für Zahlen ist etwa `Number`. Wenn Sie versuchen, einen String oder sonstigen Wert in einer Variablen dieses Typs zu speichern, dann gibt es eine Fehlermeldung. Die folgende Version des vorigen Beispiels ist also fehlerhaft:

```
// Zahl speichern
var d:Number = 256;
// String speichern?
d = "Guten Morgen";
// --> Fehlermeldung!
```

Die zugehörige Fehlermeldung wird im Ausgabefenster angezeigt und sieht so aus:

```
**Fehler** Szene=Szene 1, Ebene=Ebene 1, Bild=1:Zeile 4:
 Typenkonflikt in Zuweisungsanweisung: String gefunden, aber Number wird benötigt.
        d = "Guten Morgen";
```



Wenn Sie die Typisierung benutzen, dann müssen beim Exportieren der Flash Player 6 oder 7 und ActionScript 2.0 aktiviert sein. Das Fatale ist, dass Flash andernfalls nicht etwa einfach den Export verweigert, sondern eine SWF-Datei speichert, die sich unvorhersagbar seltsam verhält.

Neben `Number` besitzen auch die anderen oben vorgestellten einfachen Arten von Literalen ihre passenden Datentypen: Für Strings steht der Datentyp `String` zur Verfügung, für Boolesche Werte der Typ `Boolean`. Die zahlreichen anderen Datentypen von ActionScript 2.0 sind nicht für einfache Variablen gedacht, sondern für den Zugriff auf komplexe Objekte wie Datum und Uhrzeit oder gar grafische Bedienelemente.

Die wichtigste Frage ist aber, welchen Vorteil die Typisierung hat, denn zunächst einmal sieht sie wie eine Einschränkung aus. Das ist auch nicht ganz falsch – aber man sollte es eher »freiwillige Selbstbeschränkung« nennen: Wenn Sie den Datentyp festlegen, zwingen Sie sich selbst zu mehr Disziplin beim Programmieren. Sie können der Variablen nur noch Werte zuweisen, die dem festgelegten Typ entsprechen, und auch nur noch Funktionen aufrufen, die einen passenden Wert zurückgeben. Auf diese Weise können Sie Fehler vermeiden beziehungsweise schneller finden.



In ActionScript ist es besonders leicht, den Wert einer Variablen auf der Bühne anzuzeigen und umgekehrt eine Benutzereingabe in einer Variablen zu speichern: Wählen Sie das Textwerkzeug aus der Werkzeugpalette, und stellen Sie in der Eigenschaftenleiste den Texttyp *Dynamischer Text* beziehungsweise *Eingabetext* ein. Unter *Var* können Sie dann den Namen einer Variablen eingeben, mit der die Textbox automatisch verknüpft wird: Der Wert, den Sie der Variablen zuweisen, erscheint als Inhalt des Textfeldes. Bei *Eingabetext* können Sie die Eingabe des Benutzers aus der Variablen auslesen. Dieses Thema wird in Kapitel 9, *Formulare erstellen*, weiter vertieft.

Arrays

Ein *Array* ist eine spezielle Variable, die nicht nur einen einzelnen Wert enthalten kann, sondern eine ganze Liste von Werten. Die Position eines Wertes im Array wird durch einen *Index* bestimmt; dies ist meistens eine Zahl, wobei das erste Element eines Arrays den Index 0 hat.

Um ein Array zu erzeugen, wird der *Konstruktor* `new Array()` aufgerufen. Schreiben Sie Folgendes, um ein leeres Array zu erhalten, das erst später mit Werten gefüllt wird:

```
var a = new Array();
```

Wenn Sie nun an bestimmten Positionen dieses Arrays Werte speichern möchten, können Sie dies durch eine normale Wertzuweisung tun; der Index steht immer in eckigen Klammern:

```
a[0] = 27;
a[1] = 45;
```

Daneben besteht auch die Möglichkeit, ein Array von vornherein mit Werten zu belegen:

```
var a = new Array ("Flash", "Dreamweaver", "Fireworks");
```

Alternativ gibt es die folgende Kurzfassung:

```
var a = ["Flash", "Dreamweaver", "Fireworks"];
```

Wenn Sie möchten, können Sie statt der Zahlen auch Schlüssel im String-Format als Indizes verwenden. Dies ermöglicht die Verwaltung datenbankähnlicher Strukturen in Arrays. Wenn Sie einzelne Elemente setzen möchten, sieht dies genauso aus wie bei numerischen Indizes:

```
var person = new Array();
person ["vorname"] = "Klaus";
person ["nachname"] = "Schmitz";
person ["geburtsdatum"] = "06.07.1965";
```

Die Syntax für eine Deklaration mit gleichzeitiger Wertzuweisung funktioniert mit solchen Schlüsseln folgendermaßen:

```
var person =
 {vorname: "Klaus", nachname: "Schmitz", geburtsdatum: "06.07.1965"};
```

Das Schönste an Arrays ist aber, dass sie als Objekte einige eingebaute Funktionen (so genannte Methoden) besitzen, mit denen Sie gezielt Elemente hinzufügen, entfernen oder sortieren können. Tabelle 6-3 zeigt die wichtigsten Array-Methoden im Überblick.

Tabelle 6-3: Die Methoden der Klasse Array

Methode	Bedeutung	Beispiel

`push (Wert)`	Element am Ende anfügen	`var a = new Array (1, 2, 3);`
		`a.push (4);`
		`// a ist nun (1, 2, 3, 4)`

`pop()`	Element am Ende entfernen und zurückgeben	`var a = new Array (1, 2, 3);`
		`var b = a.pop();`
		`// b ist 3, a ist (1, 2)`

`unshift (Wert)`	Element am Anfang einfügen	`var a = new Array (3, 5);`
		`a.unshift (1);`
		`// a ist nun (1, 3, 5)`

`shift()`	Element am Anfang entfernen und zurückgeben	`var a = new Array (2, 4, 6);`
		`var b = a.shift();`
		`// b ist 2, a ist (4, 6);`

`slice (i1, i2)`	Elemente von Index `i1` bis (ausschließlich) `i2` zurückgeben	`var a = new Array (0,2,4,6,8);`
		`var b = a.slice (3, 5);`
		`// a bleibt; b ist (4,6)`

`splice (i, l[, wert, ...])`	`l` Elemente ab Index `i` zurückgeben und entfernen; optional die angegebenen Werte einfügen	`var a = new Array (1,2,3,4,5);`
		`var b = a.splice (2,2);`
		`// a ist (1,2,5); b ist (3,4)`

`reverse()`	Array umdrehen	`var a = new Array (1,2,3,4,5);`
		`a.reverse();`
		`// a ist (5,4,3,2,1)`

`sort(Optionen)`	Array sortieren; einige Optionen werden weiter unten im Text beschrieben	`var a = new Array (2,7,5,1);`
		`a.sort();`
		`// a ist (1,2,5,7)`

`join (Trenner)`	Elemente des Arrays zu einem durch `Trenner` getrennten String verbinden	`var a = new Array (1, 3, 5);`
		`var b = a.join (" - ");`
		`// b ist "1 - 3 - 5"`

Für die Methode `sort()`, die die Elemente des Arrays sortiert, stehen zahlreiche Optionen zur Verfügung. Sie können diese Konstanten entweder numerisch oder symbolisch angeben. Als Beispiele für die Wirkung dieser Optionen werden folgende Arrays verwendet:

```
var a = new Array (100, 11, 1, 20, 12, 20);
var b = new Array ("Flash", "8", "Pro", "mit", "actionscript");
```

Wenn Sie `a.sort()` beziehungsweise `b.sort()` ohne zusätzliche Optionen aufrufen, erhalten Sie folgende Sortierreihenfolgen:

```
a.sort();
// Neuer Wert: (1, 100, 11, 12, 20, 20)
b.sort();
// Neuer Wert: ("8", "Flash", "Pro", "actionscript", "mit")
```

Die merkwürdige Position der 100 im ersten Array kommt zustande, weil jedes Array standardmäßig alphabetisch und nicht numerisch sortiert wird.

Hier nun die Konstanten mit Wirkungsbeispielen:

- `Array.CASEINSENSITIVE` oder `1`: Standardmäßig kommen alle Großbuchstaben vor jedem Kleinbuchstaben an die Reihe; wenn Sie diese Option setzen, wird die Groß- und Kleinschreibung ignoriert, Beispiel:

  ```
  b.sort (Array.CASEINSENSITIVE);
  // Neuer Wert: ("8", "actionscript", "Flash", "mit", "Pro")
  ```

- `Array.DESCENDING` oder `2`: Normalerweise wird aufsteigend sortiert; diese Option kehrt die Reihenfolge um, Beispiel:

  ```
  a.sort (Array.DESCENDING);
  // Neuer Wert: (20, 20, 12, 11, 100, 1)
  ```

- `Array.UNIQUESORT` oder `4`: Das Array wird nur sortiert, wenn es lauter einmalige Werte enthält. Andernfalls wird 0 zurückgegeben, Beispiele:

  ```
  a.sort (Array.UNIQUESORT);
  // Wert von a bleibt unverändert
  b.sort (Array.UNIQUESORT);
  // Neuer Wert: ("8", "Flash", "Pro", "actionscript", "mit");
  ```

- `Array.RETURNINDEXEDARRAY` oder `8`: Das Array selbst wird nicht sortiert, stattdessen wird ein Array mit den Indizes (nicht den Elementen) in Sortierreihenfolge als Wert zurückgegeben, Beispiel:

  ```
  var c = b.sort (Array.RETURNINDEXEDARRAY);
  // Wert von b bleibt unverändert
  // Wert von c: (2, 0, 1, 4, 3)
  ```

- `Array.NUMERIC` oder `16`: Das Array wird numerisch statt alphabetisch sortiert, Beispiel:

  ```
  a.sort (Array.NUMERIC);
  // Neuer Wert: (1, 11, 12, 20, 20, 100);
  ```

Wenn Sie mehrere Optionen verwenden möchten, können sie diese durch bitweise Oder (|) kombinieren (oder auch addieren, was aber langsamer ist). Das folgende Beispiel sortiert das Array `a` numerisch und absteigend:

```
a.sort (Array.DESCENDING | Array.NUMERIC);
// Neuer Wert: (100, 20, 20, 12, 11, 1);
// Alternative Schreibweisen: a.sort (2 | 16) oder gar a.sort (18)
```

Beachten Sie, dass es noch weitere Syntaxformen von `sort()` gibt, die aber den Rahmen dieses Buches sprengen würden.

Ein letzter interessanter Bestandteil eines Arrays ist seine Eigenschaft `length`, die angibt, wie viele Elemente das Array enthält. Im obigen Beispiel würde `a.length` den Wert 6 zurückliefern.

Operationen

Um Literale und Variablen zu komplexen Ausdrücken zu verknüpfen, werden *Operatoren* verwendet. Viele dieser Zeichen für Verknüpfungsvorschriften kennen Sie im Alltag aus der Mathematik: `3 + 7` besagt, dass die Werte 3 und 7 addiert werden sol-

len, so dass der gesamte Ausdruck den Wert 10 besitzt. Dass einige Operatoren stärker binden als andere und deshalb in einer bestimmten Reihenfolge berechnet werden, kennen Sie sicher ebenfalls (»Punkt- vor Strichrechnung«): `3 + 7 * 5 + 4` hat den Wert 42, weil `7 * 5` zuerst ausgerechnet wird. Diese Operationen funktionieren in ActionScript genauso wie in der Mathematik. Selbst der Einsatz von Klammern zur Änderung der Priorität ist möglich: `(3 + 7) * (5 + 4)` ergibt 90.



In der Mathematik benutzt man bei der Verschachtelung mehrerer Klammern manchmal runde und eckige Klammern gemischt. In ActionScript und anderen Programmiersprachen dürfen Sie für diese Aufgabe nur runde Klammern benutzen; eckige erfüllen eine andere Aufgabe.

Die bekannteste Art von Operatoren sind somit die *arithmetischen*. ActionScript kennt die vier Grundrechenarten Addition (+), Subtraktion (-), Multiplikation (*) und Division (`/`). Sie funktionieren wie erwartet; Division durch 0 ist wie in der Mathematik verboten. Zusätzlich gibt es den so genannten Modulo-Operator (geschrieben `%`), der den Rest einer Division der beiden Werte zurückgibt: `4 % 2` ergibt beispielsweise 0, während `11 % 4` das Ergebnis 3 liefert.

Der Operator + hat übrigens noch eine andere Aufgabe: Er kann Strings miteinander verknüpfen. Beispiel:

```
var prog = "Flash";
var version = "8 Pro";
var ausgabe = prog + " " + version;
```

`ausgabe` hat nun natürlich den Wert `"Flash 8 Pro"`. Übrigens funktioniert die Verknüpfung auch dann, wenn einzelne Operanden keine Strings sind – sie werden automatisch umgewandelt:

```
var zahl = 7;
var wert = "Die Zahl ist " + zahl;
```

Hier hat `wert` den Inhalt `"Die Zahl ist 7"`.

Die nächste Gruppe sind die *Vergleichsoperatoren*. Sie vergleichen den Wert von Ausdrücken miteinander und liefern true oder false. Tabelle 6-4 zeigt die Vergleichsoperatoren mit Beispielen.

Tabelle 6-4: Die ActionScript-Vergleichsoperatoren im Überblick

Operator	Bedeutung	Beispiele
`==`	Gleichheit	`2 == 3` ergibt `false` `2 == 2` ergibt `true`

`===`	Strikte Gleichheit (Datentyp muss übereinstimmen)	`2 === 2` ergibt `true`
		`2 === "2"` ergibt `false`
		`2 === 3` ergibt `false`

Tabelle 6-4: Die ActionScript-Vergleichsoperatoren im Überblick (Fortsetzung)

Operator	Bedeutung	Beispiele
`!=`	Ungleichheit	`2 != 3` ergibt `true`
		`2 != 2` ergibt `false`
`!==`	Strikte Ungleichheit (es genügt, wenn die Datentypen verschieden sind)	`2 !== 2` ergibt `false`
		`2 !== '2'` ergibt `true`
		`2 !== 3` ergibt `true`
`<`	kleiner als	`2 < 3` ergibt `true`
		`2 < 2` ergibt `false`
`>`	größer als	`2 > 3` ergibt `false`
		`3 > 2` ergibt `true`
`<=`	kleiner oder gleich	`2 <= 3` ergibt `true`
		`2 <= 2` ergibt `true`
		`3 <= 2` ergibt `false`
`>=`	größer oder gleich	`2 >= 3` ergibt `false`
		`2 >= 2` ergibt `true`
		`3 >= 2` ergibt `true`

Vergleichsoperatoren funktionieren übrigens auch mit Strings. Hier gilt, dass ein String »kleiner als« ein anderer ist, wenn das erste unterschiedliche Zeichen weiter vorn im Zeichensatz steht. Daraus ergeben sich einige Faustregeln:

- Ziffern und die meisten Satzzeichen sind »kleiner als« Buchstaben.
- Großbuchstaben sind »kleiner als« Kleinbuchstaben.
- Umlaute und andere diakritische Zeichen sind »größer als« *alle* Buchstaben.
- Wenn Sie einen bestehenden String nach rechts verlängern, ist die längere Fassung immer »größer als« die kürzere.

Daraus ergeben sich etwa folgende Beispiele:

- `"ABC" < "abc"` ergibt `true`
- `"abc" < "ÄÖÜ"` ergibt `true`
- `"123" < "ABC"` ergibt `true`
- `"a" < "Z"` ergibt `false`
- `"Köln" < "König"` ergibt `true`
- `"Köln" < "Kölner"` ergibt `true`

Die nächste wichtige Gruppe sind die *logischen Operatoren*. Sie werden oft benutzt, um Vergleichsoperationen miteinander zu verknüpfen. Die folgenden drei Operatoren sind definiert:

- *Logisches Und*: `Ausdruck1 && Ausdruck2` ist nur wahr, wenn *beide* Ausdrücke wahr sind. Beispielsweise hat `2 < 3 && 4 > 3` den Wert `true`, während `2 <= 7 && 3 > 4` das Ergebnis `false` hat.
- *Logisches Oder*: `Ausdruck1 || Ausdruck2` ist wahr, wenn *mindestens einer* der beiden Ausdrücke wahr ist.[3] `2 < 3 || 3 > 4` hat also den Wert `true`, während `3 == 4 || 4 >= 5` das Ergebnis `false` liefert.
- *Logisches Nicht*: Wenn Sie einem Ausdruck ein `!` voranstellen, wird er verneint – `true` und Zahlenwerte außer 0 werden `false`, während `false` und 0 den Wert `true` annehmen. `!(3 < 2)` ist also beispielsweise `true`.

Die nächste Gruppe sind die so genannten *Bit-Operatoren*. Sie arbeiten zum Teil wie die logischen Operatoren, betrachten dabei aber jedes einzelne Bit eines gespeicherten Wertes.[4] Hier die Bit-Operatoren im Überblick:

- *Bitweise Und* (`&`) setzt alle Bits auf 1, die in *beiden* verknüpften Werten 1 sind. Beispielsweise ergibt `45 & 62` den Wert 44. Verstehen können Sie dies am besten, wenn Sie die Werte in die Binärdarstellung umwandeln: `101101 & 111110 = 101100`.
- *Bitweise Oder* (`|`) setzt diejenigen Bits auf 1, die in *mindestens einem* der beiden verknüpften Zahlen den Wert 1 haben. Beispiel: `45 | 62 = 63` (Binärdarstellung: `101101 | 111110 = 111111`).
- *Bitweise XOR* (Exklusiv-Oder, Zeichen `^`) bedeutet »Entweder-Oder«: Hier werden nur diejenigen Bits auf 1 gesetzt, die in *genau einem* der verknüpften Werte 1 sind. Das übliche Beispiel: `45 ^ 62 = 19` (`101101 ^ 111110 = 010011`).
- *Bitweise Nicht* (`~`) verneint jedes einzelne Bit eines Wertes. Beispiel: `~45 = -46`. Dieses merkwürdige Ergebnis liegt an der speziellen Art und Weise, wie positive und negative Zahlen gespeichert werden: Das so genannte *Zweierkomplement* setzt bei einer negativen Zahl das vorderste Bit (Vorzeichenbit) auf 1, negiert die restlichen Bits der entsprechenden positiven Zahl und addiert 1 dazu. Bei angenommenen 8 Bit ergäbe dies für das Beispiel 45: `~00101101 = 11010010`.
- Die *Bitverschiebung nach links* (`<<`) verschiebt einen Wert um die angegebene Anzahl von Bits nach links. Dies entspricht im Grunde einer mehrfachen Multiplikation mit 2. Beispielsweise ergibt `45 << 3` den Wert 360, entspricht also der Operation `45 * 8`.

---

3 Das Zeichen | wird unter Windows mit **Alt Gr** + **<** erzeugt, auf dem Mac mit **ALT** + **7**.

4 Genaue Details darüber, wie Daten im Computer gespeichert werden, würden hier zu weit führen. Sie finden sie zusammen mit vielen anderen IT-Informationen unter *http://www.galileocomputing.de/openbook/kit* – dies ist die Online-Version meines Buches »Kompendium der Informationstechnik«.

- Entsprechend erwirkt die *Bitverschiebung nach rechts* (>>), dass der Wert entsprechend oft durch 2 geteilt wird, wobei die Bits nach rechts einfach verschwinden. `45 >> 2` ergibt also beispielsweise 11.
- Eine *Variante* der Verschiebung nach rechts (>>>) füllt die freien Stellen mit Nullen auf. Der Unterschied macht sich schnell bei negativen Zahlen bemerkbar: `-45 >>> 2` ergibt 1073741812, während `-45 >> 2` das Ergebnis -12 hat.

Nun fragen Sie wahrscheinlich nach dem praktischen Nutzen der Bit-Operatoren. Sie werden sehr häufig in der Kryptographie (Ver- und Entschlüsselung, digitale Signatur und so weiter) angewendet. Bekannte Krypto-Verfahren wie MD5 oder 3DES basieren unter anderem auf solchen Operationen.

Hier eine Flash-typischere Anwendung: Die weiter unten vorgestellte Klasse `Color`, die Sie zum Einfärben von Movieclips verwenden können, benötigt den RGB-Farbwert als 32-Bit-Integer. Sie können ihn hexadezimal angeben, was den sofortigen Einsatz von HTML-Farbcodes ermöglicht – beispielsweise ist `0xFF9900` orange. Alternativ lässt sich ein kompletter Farbwert aber auch aus seinen dezimalen Rot-, Grün- und Blau-Einzelwerten berechnen. Verwenden Sie dazu den folgenden Ausdruck:

```
rotwert << 16 | gruenwert << 8 | blauwert
```

Das ist praktischer (und wird etwas schneller berechnet) als die ebenfalls gültige arithmetische Fassung:

```
rotwert * 65536 + gruenwert * 256 + blauwert
```

Für die Wertzuweisung an Variablen existieren neben dem einfachen `=` noch eine Reihe von *Modifikationsoperatoren*, die den bisherigen Wert einer Variablen um einen bestimmten Wert verändern. Betrachten Sie beispielsweise folgende Anweisung:

```
x = x + 1;
```

Anders als in der Mathematik ergibt dies keinen unauflösbaren Widerspruch, weil der Ausdruck auf der rechten Seite bei einer Wertzuweisung immer zuerst komplett berechnet wird. Als neuer Wert von `x` wird also der um 1 erhöhte alte Wert festgelegt.

Einige Beispiele für Modifikatoren finden Sie zusammen mit der zugehörigen Langfassung in Tabelle 6-5.

Tabelle 6-5: Modifikationsoperatoren, die den Wert von Variablen verändern

Langfassung	Kurzfassung	Erläuterung
`x = x + 3;`	`x += 3;`	Variable um den angegebenen Wert erhöhen

`x = x + 1;`	`x += 1;`
	`x++;`
	`++x;`
		Variable um 1 erhöhen
		(Unterschied zwischen `x++` und `++x` wird im Text erläutert)

`x = x - 3;`	`x -= 3;`	Variable um den angegebenen Wert vermindern

Tabelle 6-5: Modifikationsoperatoren, die den Wert von Variablen verändern (Fortsetzung)

Langfassung	Kurzfassung	Erläuterung

`x = x - 1;`	`x -= 1;`
	`x--;`
	`--x;`
		Variable um 1 vermindern

`x = x * 3;`	`x *= 3;`	Variable mit dem angegebenen Wert multiplizieren
`x = x / 3;`	`x /= 3;`	Variable durch den angegebenen Wert dividieren

Für die Erhöhung beziehungsweise Verminderung um 1 stehen die speziellen Operatoren `++` beziehungsweise `--` zur Verfügung. Sie können sie sowohl vor als auch hinter die gewünschte Variable schreiben; bei einer allein stehenden Anweisung macht dies keinen Unterschied:

```
a++;
```

erhöht den Wert von `a` genauso um 1 wie

```
++a;
```

Die beiden Formulierungen verhalten sich allerdings unterschiedlich, wenn Sie sie innerhalb eines komplexeren Ausdrucks verwenden: Das vorangestellte `++` (*Prä-Inkrement*) erhöht die Variable zuerst um 1 und verwendet dann im Ausdruck den neuen Wert:

```
var a = 3;
b = ++a;     // a: 4, b: 4
```

Ein nachgestelltes `++` (*Post-Inkrement*) verwendet dagegen den bisherigen Wert im Ausdruck und erhöht die Variable erst danach um 1:

```
var a = 3;
b = a++;    // a: 4; b: 3
```

Für `--` (*Prä-* beziehungsweise *Post-Dekrement*) gilt sinngemäß das Gleiche.

Neben den hier erwähnten Modifikationsoperatoren gibt es noch weitere, nämlich für sämtliche logischen Operatoren und Bit-Operatoren. Beispiele: `&&=` oder `|=`.

Ein letzter interessanter Operator ist das *ternäre* (dreigliedrige) `Ausdruck1 ? Ausdruck2 : Ausdruck3`. Es handelt sich um eine *Wenn-Dann-Operation*: Wenn `Ausdruck1` wahr ist, wird `Ausdruck2` gewählt, ansonsten `Ausdruck3`. Beispiel:

```
3 < 4 ? 1 : 0
```

Dies liefert das Ergebnis 1, weil 3 tatsächlich kleiner als 4 ist.

Zum Abschluss des Themas Operatoren sehen Sie hier deren *Rangfolge*. Je weiter oben ein Operator in der Liste steht, desto stärker bindet er und desto früher wird er ausgewertet:

- `()` – Klammern zur gezielten Erhöhung der Rangfolge
- `!`, `~`, `++`, `--`, `+` (Vorzeichen), `-` (Vorzeichen)
- `*`, `/`, `%`

- `+` und `-` (arithmetische Operatoren)
- `<<`, `>>` und `>>>` (Bit-Verschiebung)
- `<`, `<=`, `>` und `>=`
- `==`, `!=`, `===` und `!==`
- `&` (bitweise Und)
- `^` (bitweise Exklusiv-Oder)
- `|` (bitweise Oder)
- `&&` (logisches Und)
- `||` (logisches Oder)
- `?:` (Wenn-Dann-Operator)
- `=`, `+=`, `-=` usw. (Zuweisungsoperatoren)

Kontrollstrukturen

In einem Computerprogramm ist es wichtig, die so genannte *Flusskontrolle* durchzuführen: Beim Auftreten bestimmter Bedingungen muss das Programm an eine andere Stelle verzweigen oder bestimmte Anweisungen erneut ausführen. In ActionScript stehen für die Flusskontrolle, wie in vielen anderen Sprachen, *Fallentscheidungen* und *Schleifen* zur Verfügung.

Fallentscheidungen

Mit Hilfe einer Fallentscheidung kann ein Programm eine Bedingung prüfen. Falls die Bedingung zutrifft, werden bestimmte Anweisungen ausgeführt; wenn sie dagegen falsch ist, können andere (oder auch keine) Befehle ausgeführt werden.

Die einfachste Form der Fallentscheidung hat das folgende Format:

```
if (Ausdruck)
   Anweisung;
```

Falls der *`Ausdruck`* gemäß den oben erläuterten Kriterien wahr ist, wird die zugehörige Anweisung ausgeführt. Im folgenden Beispiel erfolgt ein Sprung zum Bild *ende*, wenn die Variable `spiele` den Wert 0 hat:

```
if (spiele == 0)
   gotoAndStop ("ende");
```

Wie bereits erwähnt, können Sie einen Block in geschweiften Klammern benutzen, wenn mehrere Anweisungen von derselben Bedingung abhängen sollen. Betrachten Sie dazu dieses Beispiel, das der Variablen `spiele` vor dem Sprung zum Bild *ende* den Wert 3 zuweist (etwa als Ausgangspunkt für eine neue Spielrunde):

```
if (spiele == 0) {
   spiele = 3;
   gotoAndStop ("ende");
}
```

Wenn eine alternative Anweisung ausgeführt werden soll, falls der Ausdruck *nicht* wahr ist, geschieht dies mit Hilfe des Schlüsselwortes `else`. Das folgende Beispiel könnte am Ende einer Spielrunde stehen: Die Variable `spiele` wird um 1 vermindert. Wenn ihr Wert noch größer als 0 ist, erfolgt ein Sprung zum Frame *spielstart*; andernfalls verzweigt der Film zum Bild *ende*:

```
spiele--;
if (spiele > 0) {
   gotoAndPlay ("spielstart");
} else {
   gotoAndStop ("ende");
}
```

Natürlich könnten Sie auch bei einem `else`-Teil die geschweiften Klammern weglassen, wenn er nur eine Anweisung umfassen soll.

In manchen Fällen erfolgt die Überprüfung einer weiteren Bedingung, falls die erste Bedingung nicht zutrifft. So entsteht ein verschachtelter Block in der Form `if` – `else if` – ... – `else`. Das folgende Beispiel nimmt bei unterschiedlichen Punktzahlen in einem Spiel verschiedene Wertungen vor:

```
if (punkte < 100) {
   bewertung = "Sie sollten noch üben";
} else if (punkte < 200) {
   bewertung = "Na ja";
} else if (punkte < 500) {
   bewertung = "Schon ganz nett";
} else {
   bewertung = "Sie haben geschummelt, oder?!";
}
```

Da die Überprüfungen hier einen Fall nach dem anderen ausschließen, können Sie sich beim letzten `else` ein weiteres `if` sparen – hier kann `punkte` nur größer oder gleich 500 sein.

Wenn Sie keine Wertebereiche, sondern eine Reihe verschiedener Einzelwerte überprüfen möchten, können Sie auch die nützliche `switch`/`case`-Fallentscheidung verwenden. Ihre grundlegende Syntax sieht so aus:

```
switch (Variable) {
case Wert1 :
   Anweisung(en);
   break;
case Wert2 :
   Anweisung(en);
   break;
   ...
default:
   Anweisung(en);
}
```

Die Variable wird jeweils mit den einzelnen case-Werten verglichen. Falls sie mit einem dieser Werte übereinstimmt, wird der Block ab dieser Stelle ausgeführt. Standardmäßig werden alle Anweisungen bis zum } Ende des Blocks abgearbeitet; da dies meist nicht gewünscht ist, steht in der Regel vor dem jeweils nächsten case ein break, das den Block verlässt. Der (optionale) Einstiegspunkt default wird gewählt, wenn die Variable *keinem* der angegebenen Werte entspricht. Er gilt also für »alle anderen Werte« und entspricht damit dem letzten else in einer verschachtelten if/else-Abfolge. Daher ist default beispielsweise nützlich, um unzulässige Werte (etwa Benutzereingaben) abzufangen.

Das folgende Beispiel vergleicht den Wert der Variablen auswahl mit verschiedenen Zeichen:

```
switch (auswahl) {
case "a" :
   gotoAndPlay ("anfang");
   break;
case "s" :
   gotoAndPlay ("spiel");
   break;
case "e" :
case "q" :
   gotoAndStop ("ende");
default :
   gotoAndPlay ("ungueltig");
}
```

Der Wert von auswahl entspricht hier offensichtlich einem Tastendruck zur Steuerung einer Flash-Anwendung. Interessant sind die beiden aufeinander folgenden case-Zeilen für die Werte "e" und "q": Da erst das nächste break den Block wieder verlässt, können Sie auf diese Weise für verschiedene Elemente dieselben Anweisungen ausführen.

Schleifen

Es kommt oft vor, dass ein Programm bestimmte Aufgaben mehrmals erledigen soll. Natürlich können Sie in Flash stets mit gotoAndPlay() zu einem früheren Bild zurückspringen, um eine solche Wiederholung in der Zeitleiste einzurichten. Wenn es aber speziell darum geht, ActionScript-Anweisungen zu wiederholen, sollten Sie eher eine *programmierte Schleife* benutzen. Davon gibt es verschiedene Typen, die hier kurz vorgestellt werden.

Die einfachste Sorte von allen ist die while-Schleife. Genau wie eine if-Fallentscheidung überprüft sie zunächst eine Bedingung, bevor sie die von ihr abhängige Anweisung (oder den Anweisungsblock) ausführt. Nach der Ausführung wird die Bedingung allerdings erneut überprüft – und dies so lange, bis die Bedingung falsch ist. Das folgende Beispiel gibt die Zahlen von 1 bis 10 im Ausgabefenster aus:

```
var i = 0;
while (i < 10) {
   i++;
   trace ("Bin jetzt bei " + i);
}
```

Da die Bedingung überprüft wird, *bevor* die abhängigen Anweisungen (der so genannte *Schleifenrumpf*) ausgeführt werden, wird die `while`-Anweisung als *kopfgesteuerte* Schleife bezeichnet. Wenn die Bedingung von Anfang an nicht zutrifft, kann es vorkommen, dass der Schleifenrumpf gar nicht ausgeführt wird.

Das entgegengesetzte Modell, die *fußgesteuerte* Schleife, wird dagegen immer mindestens einmal ausgeführt, weil die Bedingung erst nach der Ausführung geprüft wird. Sie besitzt folgende Syntax:

```
do {
   Anweisung(en);
} while (Ausdruck);
```

Fußgesteuerte Schleifen sind immer dann nützlich, wenn die zu prüfende Bedingung sich erst durch den Schleifenrumpf selbst ergibt – beispielsweise bei der Überprüfung von Eingaben. Das folgende Beispiel zählt, wie lange es dauert, eine 6 zu »würfeln«:

```
var wurf;
// Bisher 0 Versuche
var versuche = 0;
// Würfeln, bis die 6 kommt
do {
   versuche++;
   // Würfelwurf
   wurf = Math.ceil (Math.random() * 6);
} while (wurf != 6);
trace ("Es wurden " + versuche + " Würfe benötigt.";
```

Die letzte Möglichkeit, eine Schleife zu konstruieren, ist die `for()`-Schleife. Sie besitzt folgendes Format:

```
for (Initialisierung; Bedingung; Wertänderung) {
   Anweisung(en);
}
```

Die *`Initialisierung`* und die *`Wertänderung`* können beliebige Anweisungen sein, die *`Bedingung`* ein beliebiger Ausdruck. Die Initialisierung wird vor dem ersten Durchlauf ausgeführt, die Bedingung wird vor Beginn jedes Durchlaufs überprüft, und die Wertänderung findet nach jedem Durchlauf statt. `for`-Schleifen werden meist benutzt, um Schleifen mit einer feststehenden Anzahl von Durchläufen zu formulieren. Das folgende Beispiel gibt nacheinander die Werte 2, 4, 6, 8 und 10 im Ausgabefenster aus:

```
for (var i = 2; i <= 10; i += 2) {
   trace (i);
}
```

Die Klasse Math – mathematische Funktionen

Die ECMAScript-Basisklasse `Math` enthält zahlreiche Funktionen und Konstanten der Mathematik, die für die ActionScript-Programmierung oft sehr nützlich sind. Im obigen Beispiel wurden der Zufallsgenerator `Math.random()` sowie die Aufrundungsfunktion `Math.ceil()` verwendet. `Math.random()` liefert eine Fließkomma-Zufallszahl (genauer gesagt eine nicht vorhersagbare Zahl; echten Zufall kennt ein Computer nicht) zwischen 0 und 1. Um mit ihrer Hilfe einen Würfelwurf zu simulieren, wird sie zunächst mit 6 multipliziert – das Ergebnis ist eine Fließkommazahl zwischen 0 und (etwas weniger als) 6. Durch `Math.ceil()` wird jeweils die nächsthöhere Ganzzahl gewählt. Diese liegt aufgrund der vorherigen Multiplikation im gewünschten Bereich zwischen 1 und 6.

Hier einige weitere Funktionen und Konstanten der Klasse `Math`:

- `Math.floor()` rundet eine Fließkommazahl auf die nächstniedrigere Ganzzahl ab: `Math.floor (5.3)` wird 5, aber `Math.floor (5.8)` ebenfalls.
- `Math.round()` rundet dagegen mathematisch korrekt: `Math.round (5.3)` ergibt 5, während `Math.round (5.6)` den Wert 6 hat.
- `Math.abs()` liefert den *Betrag* eines Wertes, entfernt also das Vorzeichen: `Math.abs (64)` ist 64, `Math.abs (-64)` hat ebenfalls das Ergebnis 64.
- `Math.sin()`, `Math.cos()` und `Math.tan()` liefern den Sinus, Cosinus beziehungsweise Tangens eines Wertes. Vorsicht: Sie arbeiten im *Bogenmaß*. Dies ist ein Kreisabschnitt auf dem Einheitskreis statt eines Winkels; der ganze Kreis hat nicht den Wert 360, sondern 2 * π.
- `Math.pow()` errechnet Potenzen: Math.pow (2, 8) entspricht 2 hoch 8 und ergibt damit 256.
- `Math.PI` und `Math.E` sind die mathematischen Konstanten π beziehungsweise e (Eulersche Zahl, Basis des natürlichen Logarithmus).

Es ist kein Problem, diese (und jede andere) `for`-Schleife in eine `while`-Schleife umzuwandeln:

```
var i = 2;
while (i <= 10) {
   trace (i);
   i += 2;
}
```

Hier als Abschluss des Themas Schleifen noch ein kleines Beispiel: Es zeigt im Ausgabefenster sechs zufällig gezogene Lottozahlen sowie eine Zusatzzahl an. Dazu wird ein Array mit allen 49 Zahlen (»Kugeln«) angelegt. Mit Hilfe der Funktion splice() wird jeweils eine zufällige Kugel aus diesem Array entfernt und in einem zweiten Array gespeichert:

```
// Arrays für Kugelvorrat und gezogene Kugeln einrichten
var kugeln = new Array();
var gezogen = new Array();
// Die 49 Kugeln erzeugen und im Array speichern
for (var i = 1; i <= 49; i++) {
   kugeln.push(i);
}
// Sieben Kugeln "ziehen"
for (var j = 0; j < 7; j++) {
   // Zufallszahl zwischen 1 und der restlichen Kugelanzahl
   var z = Math.floor (Math.random() * kugeln.length);
   // Element an der Zufallsposition entfernen
   var aktuell = kugeln.splice (z, 1);
   // Gezogenes Element speichern
   gezogen.push (aktuell);
}
// Ausgabe
for (i = 0; i < 6; i++) {
   trace (gezogen [i]);
}
trace ("Zufallszahl: " + gezogen [6]);
```

Funktionen

Wenn Sie bestimmte Codeblöcke immer wieder benötigen, bietet es sich an, sie in eine *Funktion* auszulagern. Im Grunde sind Funktionen nichts weiter als benannte Anweisungsfolgen. Sie haben in diesem Kapitel bereits zahlreiche vordefinierte Funktionen kennen gelernt, zum Beispiel `trace()` oder `gotoAndPlay()`. Wenn Sie selbst Funktionen definieren, können Sie sie innerhalb Ihres Films selbstverständlich genauso einsetzen.

Setzen Sie Funktionsdefinitionen einfach in ein Bildskript; am einfachsten in das erste Schlüsselbild der gewünschten Zeitleiste. Sie werden durch das Schlüsselwort `function` eingeleitet und umschließen grundsätzlich einen Anweisungsblock. Das folgende Beispiel zeigt eine Funktion, die nichts weiter tut, als ihren eigenen Aufruf (im Ausgabefenster) zu bestätigen:

```
function test()
{
   trace ("test() wurde aufgerufen!");
}
```

Geben Sie nun an einer anderen Stelle in Ihrem Code folgende Anweisung ein:

```
test();
```

Dies ist ein Aufruf von `test()`, der die erwähnte Ausgabe erzeugt.

Noch nützlicher sind allerdings Funktionen, die Werte entgegennehmen. Zu diesem Zweck können Sie in den Klammern hinter dem Funktionsnamen so genannte *Parametervariablen* einsetzen. Das folgende Beispiel gibt die größere von zwei übergebenen Zahlen aus:

```
function groesser (z1, z1)
{
   if (z1 > z2) {
      trace (z1 + " ist groesser");
   } else if (z2 > z1) {
      trace (z2 + " ist groesser");
   } else {
      trace ("Die Zahlen sind gleich groß.");
   }
}
```

Diese Funktion wird natürlich mit zwei Argumenten aufgerufen:

```
groesser (5, 7);
```

liefert die Ausgabe »7 ist groesser«.

Umgekehrt kann eine Funktion auch einen Wert zurückgeben, so dass Sie ihren Aufruf in einen Ausdruck einsetzen können. Dazu wird das Schlüsselwort `return` verwendet, das die Funktion sofort verlässt und den entsprechenden Rückgabewert zurückliefert. Das folgende Beispiel gibt `true` zurück, wenn die übergebene Zahl gerade (durch zwei teilbar) ist, ansonsten `false`:

```
function istGerade (zahl)
{
   if (zahl % 2 == 0) {
      return true;
   }
   // Wenn wir noch da sind, war sie ungerade!
   return false;
}
```

Wie Sie sehen, braucht diese Funktion kein `else` – die Tatsache, dass das `return` noch nicht stattgefunden hat und dass die Funktion deshalb weiter ausgeführt wird, bedeutet bereits, dass die Bedingung falsch war.

Sie können die Funktion nun zum Beispiel zur Bedingungsprüfung benutzen:

```
if (istGerade (b)) {
   trace (b + " ist gerade!");
}
```

Hier noch ein weiteres praktisches Beispiel, das Sie etwa zur Überprüfung von Benutzereingaben einsetzen können – es testet, ob der übergebene Wert eine Zahl ist oder nicht:

```
function istZahl (wert)
{
   return !isNaN (wert);
}
```

Die Funktion macht es sich sehr einfach: Sie gibt das Gegenteil der eingebauten Funktion `isNaN()` zurück (»is not a number«). Diese liefert `true`, wenn das Argument keine Zahl ist, und andernfalls `false`. Durch das vorangestellte `!` wird dieses Verhalten umgekehrt.

Beachten Sie, dass auch Funktionen und ihre Parameter feste Datentypen besitzen können. Die folgende ActionScript 2.0-Variante von `istGerade()` akzeptiert ausschließlich Zahlen und besitzt selbst den Datentyp `Boolean`:

```
function istGerade (zahl:Number):Boolean
{
   if (zahl % 2 == 0) {
      return true;
   }
   return false;
}
```

Movieclips steuern

Die besondere Leistungsfähigkeit von ActionScript ergibt sich erst aus der Tatsache, dass jeder Movieclip eine unabhängige Zeitleiste besitzt. Auf diese Weise kann ein einzelner Film beliebig viele unabhängig »handelnde Subjekte« enthalten. Jede separate Zeitleiste – also jeder Hauptfilm und jedes Movieclip-Symbol – kann dabei Bildaktionen enthalten. Darüber hinaus können die unterschiedlichen Zeitleisten sich aber auch gegenseitig steuern; dies ist wahrscheinlich die interessanteste Fähigkeit von ActionScript.

Erstellen Sie zur Einstimmung in die Thematik folgendes kleine Beispielszenario:

1. Erzeugen Sie zwei Schaltflächen-Symbole mit den Beschriftungen *Stop* und *Weiter*.
2. Erstellen Sie ein neues Movieclip-Symbol mit der Bezeichnung *quer*. Richten Sie innerhalb dieses Symbols eine einfache Animation ein, bei der sich ein Objekt seitlich hin und her bewegt. Legen Sie die beiden Schaltflächen auf eine zweite Ebene. Weisen Sie dem Button *Stop* folgendes Skript zu:

   ```
   on (release) {
      stop();
   }
   ```

   Die Schaltfläche *Weiter* erhält folgenden ActionScript-Code:

   ```
   on (release) {
      play();
   }
   ```

3. Erzeugen Sie auf der Bühne des Hauptfilms eine weitere Animationssequenz, in der sich ein Objekt von oben nach unten und wieder zurück bewegt.
4. Fügen Sie eine neue Ebene hinzu. Ziehen Sie den Movieclip *quer* auf die Bühne.
5. Ziehen Sie die beiden Schaltflächen auch in den Hauptfilm (natürlich so, dass sie den entsprechenden Schaltflächen innerhalb der Movieclip-Instanz nicht in die Quere kommen). Weisen Sie ihnen hier dieselben Skripte zu wie unter Punkt 2.

Wenn Sie nun den Befehl *Film testen* ausführen, werden Sie feststellen, dass Sie die beiden Animationen unabhängig voneinander steuern können – die Zeitleiste der Movieclip-Instanz und die des Hauptfilms werden separat durch Schaltflächen gesteuert.

Die Movieclip-Hierarchie

Damit Movieclips überhaupt von außen angesprochen werden können, benötigen sie einen *Instanznamen*. Dieser wird links oben in der Eigenschaftenleiste eingetragen. Achten Sie darauf, dass Sie sich auch bei Instanznamen an die weiter oben beschriebenen Regeln für Bezeichner halten.

Wenn der Hauptfilm Movieclip-Instanzen enthält, deren zugrunde liegende Symbole ihrerseits ebenfalls Instanzen anderer Symbole enthalten, ergibt sich daraus eine verschachtelte *Movieclip-Hierarchie*. Sie können von jeder Zeitleiste aus jede andere Zeitleiste ansprechen; die Herangehensweise ähnelt den verschachtelten Ordnern in einem Dateisystem.

Angenommen, auf der Bühne des Hauptfilms befindet sich eine Movieclip-Instanz namens *clip1* und eine weitere namens *clip2*. Das Symbol, auf dem *clip1* basiert, enthält wiederum eine Instanz mit dem Instanznamen *unterclip*. Schematisch ergibt sich also folgende Struktur:

```
Hauptfilm
|
+--- clip1
|    |
|    +--- unterclip
|
+--- clip2
```

Es gibt grundsätzlich zwei verschiedene Methoden, andere Zeitleisten anzusprechen: *absolute* Pfade und *relative* Pfade.

Ein absoluter Pfad geht stets vom Hauptfilm aus, dessen ActionScript-Name `_root` lautet. Der Vorteil absoluter Pfade besteht darin, dass sie aus jeder Zeitleiste heraus gleich lauten, während sie den Nachteil haben, dass sie bei tief verschachtelten Hierarchien sehr lang und unhandlich werden. Hier die absoluten Pfade aller beteiligten Filme der Beispielhierarchie:

- Hauptfilm: `_root`
- `clip1`: `_root.clip1`
- `unterclip`: `_root.clip1.unterclip`
- `clip2`: `_root.clip2`

Bei relativen Pfaden wird dagegen angegeben, wie der angesprochene Movieclip mit der aktuellen Zeitleiste verbunden ist. Die Instanznamen direkt untergeordneter

Clips können Sie einfach angeben, während der jeweils übergeordnete Film durch die Bezeichnung `_parent` angesprochen wird. Vom Hauptfilm aus lauten die relativen Pfade der drei Movieclips also wie folgt:

- `clip1`
- `clip1.unterclip`
- `clip2`

Wenn Sie innerhalb des Symbols von `clip1` Skripte verwenden, können Sie die anderen Filme daraus folgendermaßen relativ ansprechen:

- Hauptfilm: `_parent`
- `unterclip`
- `_parent.clip2`

Schließlich sehen Sie hier noch, wie Sie den Hauptfilm, `clip1` und `clip2` vom `unterclip` aus erreichen können:

- Hauptfilm: `_parent._parent`
- `clip1`: `_parent`
- `clip2`: `_parent._parent.clip2`

Es steht Ihnen übrigens frei, relative Pfade mit dem Bezeichner `this` einzuleiten. Er ermöglicht es einer Zeitleiste (oder einem anderen Objekt), formal auf sich selbst Bezug zu nehmen.

Es gibt ein sehr praktisches Hilfsmittel zum Einfügen der komplexen Movieclip-Beziehungen: Wenn Sie in der Symbolleiste der Aktionen-Palette auf das kleine Fadenkreuz klicken, öffnet sich der Dialog *Zielpfad einfügen*. Hier können Sie den anzusprechenden Film leicht aus einer Baumansicht auswählen und sich sogar entscheiden, ob die Referenz relativ oder absolut erfolgen soll. Sobald Sie *OK* drücken, wird das Ergebnis an der aktuellen Position im Skript eingetragen. Abbildung 6-10 zeigt den Dialog mit den Filmen des aktuellen Beispiels.

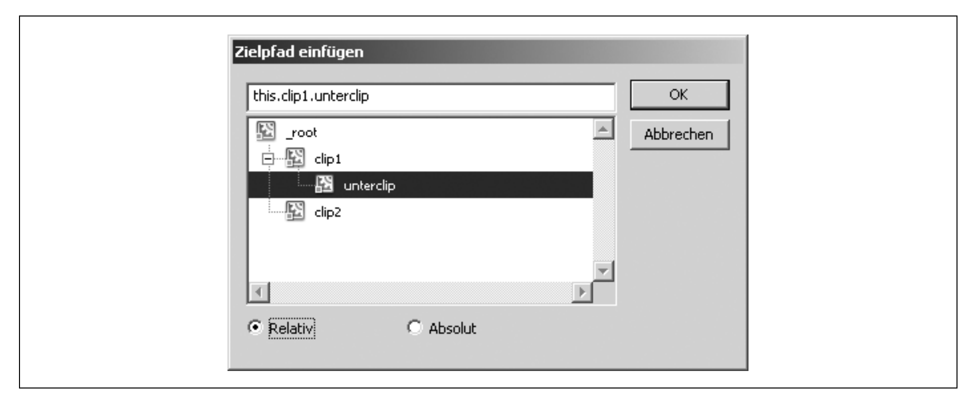


Abbildung 6-10: Der Dialog »Zielpfad einfügen« erleichtert das Ansprechen anderer Zeitleisten

Nun wissen Sie zwar bereits, wie andere Zeitleisten angesprochen werden, aber noch nicht, was Sie mit ihnen tun können. Das Wichtigste ist, dass Sie für Movieclip-Instanzen die bekannten Funktionen zur Filmsteuerung aufrufen können. In diesem Zusammenhang werden sie als *Methoden* der Movieclips bezeichnet, weil es sich um Funktionen handelt, die diesen Objekten zugeordnet sind. Die Syntax lautet grundsätzlich folgendermaßen:

```
clip.Methode (...);
```

Angenommen, Sie möchten `clip1` aus einem Skript des Hauptfilms heraus anhalten. Dies geschieht mit Hilfe der folgenden Anweisung:

```
clip1.stop();
```

Falls Sie absolute Pfade bevorzugen, lautet die Formulierung natürlich so:

```
_root.clip1.stop();
```

Möchten Sie dagegen aus dem `unterclip` heraus den Hauptfilm zu dessen Bild `"menue"` schicken, dann können Sie sich eine der beiden folgenden Anweisungen aussuchen:

```
_root.gotoAndStop ("menue");            // absolut
_parent._parent.gotoAndStop ("menue");  // relativ
```



Wenn Sie die Zeitleiste des Hauptfilms aus einem anderen Clip heraus ansprechen, können Sie in den Methoden `gotoAndStop()` beziehungsweise `gotoAndPlay()` *keine* Szenen angeben! Sie müssen sich mit der absoluten Bildnummer (von der ersten Szene an gerechnet) oder einer eindeutigen Bildmarkierung behelfen. Dieser kleine Unterschied zeigt, dass die globalen `gotoAnd*()`-Funktionen *nicht* identisch mit den gleichnamigen Movieclip-Methoden sind.

Sie können mit Hilfe der gezeigten Punkt-Syntax auch Variablen und Funktionen einer anderen Zeitleiste ansprechen. Das folgende Beispiel zeigt, wie Sie aus einem beliebigen Movieclip heraus die weiter oben vorgestellte Funktion `istZahl()` benutzen können, wenn sie in der Zeitleiste des Hauptfilms definiert ist:

```
if (_root.istZahl (a)) {
   trace (a + " ist eine Zahl!");
}
```

Das nächste Beispiel überprüft den Wert der Variablen `durchgang` in der direkt untergeordneten Instanz `clip1`:

```
if (clip1.durchgang < 3) {
   gotoAndPlay ("anfang");
} else {
   gotoAndStop ("ende");
}
```

Hier das letzte derartige Beispiel; es erhöht den Wert der Variablen zaehler im Hauptfilm von einer beliebigen Stelle aus um 1:

```
_root.zaehler++;
```



ActionScript bietet eine sehr praktische Funktion namens eval(), die einen String-Ausdruck auswertet und als ActionScript-Formulierung versteht. Dieses Hilfsmittel ermöglicht es Ihnen, dynamische Zugriffe auf Movieclip-Instanzen durchzuführen, indem Sie die Instanznamen damit bilden. Das folgende Beispiel geht davon aus, dass die aktuelle Zeitleiste 10 Instanzen namens *clip1* bis *clip10* enthält. Diese sollen in einer Schleife alle angehalten werden:

```
for (i = 1; i <= 10; i++) {
   // Referenz auf aktuellen Clip erstellen
   var clipref = eval ("clip" + i);
   // aktuellen Clip anhalten
   clipref.stop();
}
```

Movieclip-Eigenschaften

Ein weiteres sehr praktisches Element von Movieclip-Instanzen sind deren lesbare und änderbare *Eigenschaften*: Sie können Aspekte wie Größe, Position oder Transparenzgrad einer Instanz dynamisch ändern und auf diese Weise Animationen erstellen, die interaktiv auf Ereignisse reagieren.

Der Name jeder Eigenschaft beginnt mit einem Unterstrich. Tabelle 6-6 zeigt zunächst einen Überblick über die wichtigsten von ihnen.

Tabelle 6-6: Die wichtigsten Movieclip-Eigenschaften

Eigenschaft	Wertebereich	Erläuterungen
_width	beliebig	Breite in Pixeln
_height	beliebig	Höhe in Pixeln
_xscale	beliebig	Breiten-Skalierungsfaktor (anfangs 100)
_yscale	beliebig	Höhen-Skalierungsfaktor (anfangs 100)
_x	beliebig	horizontale Position
_y	beliebig	vertikale Position
_rotation	0–360	Rotationswinkel
_visible	true, false	Movieclip anzeigen / ausblenden
_alpha	0–100	Transparenzgrad (0 unsichtbar, 100 deckend)
_xmouse	beliebig	X-Position des Mauszeigers, relativ zum Registrierpunkt des Movieclips (nur lesbar!)
_ymouse	beliebig	Y-Position des Mauszeigers

Wenn Sie die Größe der Movieclip-Instanz `clip1` halbieren möchten, können Sie ihre Eigenschaften `_xscale` und `_yscale` jeweils auf 50 setzen:

```
clip1._xscale = 50;
clip1._yscale = 50;
```

Das folgende Beispiel erhöht die Breite der Movieclip-Instanz `clip2` um 10 Pixel:

```
clip2._width += 10;
```

Wenn Sie den folgenden Code einer Schaltflächen-Instanz zuweisen, erhalten Sie einen »Umschalter« für den Movieclip `_root.clip3`. Jeder Klick blendet ihn abwechselnd ein beziehungsweise aus:

```
on (release) {
   if (_root.clip3._visible) {
      _root.clip3._visible = false;
   } else {
      _root.clip3._visible = true;
   }
}
```



Im bereits erwähnten Verzeichnis */actionscript* auf der CD-ROM finden Sie den Film *eigenschaften.fla*. Er ermöglicht Ihnen die dynamische Änderung zahlreicher Eigenschaften eines Movieclips über Schaltflächen. Er erläutert nebenbei, wie Sie »Dauerfeuer«-Schaltflächen erstellen können, die nicht nur auf einen Klick reagieren, sondern beim Gedrückthalten der Maustaste weiterarbeiten.

Movieclips einfärben

Im Gegensatz zum Transparenzgrad, der sich über die Eigenschaft `_alpha` einstellen lässt, ist es ein wenig komplexer, die Farbe eines Movieclips dynamisch zu ändern: Sie müssen dazu ein `Color`-Objekt erstellen. Das Argument des `Color`-Konstruktoraufrufs ist der Bezug auf den Movieclip, dem Sie die Farbe zuweisen möchten. Das folgende Objekt wird für den Movieclip `_root.farbclip` verwendet:

```
var farbe = new Color (_root.farbclip);
```

Wenn Sie den gesamten Clip mit einer einzelnen Farbe bedecken möchten, können Sie die Methode `setRGB()` des Color-Objekts aufrufen. Diese benötigt einen RGB-Farbwert, den Sie am besten hexadezimal angeben. Das folgende Beispiel färbt die Instanz `_root.farbclip` türkis ein:

```
farbe.setRGB (0x00FF99);
```

Weiter oben bei der Beschreibung der Bit-Operatoren wurde bereits erwähnt, wie Sie diese zur Berechnung eines Farbwerts aus RGB-Einzelwerten einsetzen können. Das folgende Beispiel weist dem `farbclip` den Rotwert `rot`, den Grünwert `gruen` und den Blauwert `blau` zu; die entsprechenden Variablen benötigen Zahlenwerte zwischen 0 und 255, damit alles korrekt funktioniert:

```
farbe.setRGB (rot << 16 | gruen << 8 | blau);
```

Statt einer deckenden Farbe können Sie einer Movieclip-Instanz auch den in Kapitel 3 beschriebenen Farbeffekt *Erweitert* dynamisch zuweisen. Zu diesem Zweck enthält `Color` eine Methode namens `setTransform()`. Ihr Argument ist ein allgemeines Objekt, das acht Variablen enthalten muss:

- `ra` – Prozentwert des Rotkanals (-100 bis +100)
- `rb` – absolute Rot-Einfärbung (-255 bis +255)
- `ga` – Prozentwert des Grünkanals (-100 bis +100)
- `gb` – absolute Grün-Einfärbung (-255 bis +255)
- `ba` – Prozentwert des Blaukanals (-100 bis +100)
- `bb` – absolute Blau-Einfärbung (-255 bis +255)
- `aa` – Prozentwert des Alphakanals (-100 bis +100)
- `ab` – absoluter Alpha-Grundwert (-255 bis +255)

Wenn Sie etwa die Beispielinstanz `farbclip` etwas aufhellen (Hinzufügen von Weiß durch gleichmäßiges Erhöhen der drei Farbkanäle) und ihren Transparenzgrad auf 80% vermindern möchten, dann können Sie folgende Syntax verwenden:

```
// allgemeines Objekt für die Farbtransformation erstellen
var trans = new Object();
// die Farb- und Alphawerte setzen
trans.ra = 100;
trans.rb = 150;
trans.ga = 100;
trans.gb = 150;
trans.ba = 100;
trans.bb = 150;
trans.aa = 80;
trans.rb = 0;
// Das Color-Objekt für _root.farbclip erstellen
var farbe = new Color (_root.farbclip);
// Die Transformation einstellen
farbe.setTransform (trans);
```

Wenn Sie möchten, können Sie auch die bisherigen Farbeinstellungen einer Instanz auslesen, beispielsweise für Änderungsberechnungen: Die Methoden `getRGB()` beziehungsweise `getTransform()` liefern sie zurück. Der Farbwert von `getRGB()` ist eine Zahl zwischen 0 und etwa 16,7 Millionen – das Ergebnis der gezeigten Bit-Operation. Auf die einzelnen Werte der Transformation können Sie dagegen mittels *`colorObjekt`*`.getTransform().ra`, *`colorObjekt`*`.getTransform().rb` usw. zugreifen.

Movieclip-Instanzaktionen

Weiter oben wurde bereits erwähnt, dass Sie auch einer Movieclip-Instanz, ähnlich wie einer Schaltfläche, ein Skript zuweisen können. In einem solchen Skript wird das spezielle Konstrukt `onClipEvent()` benutzt, das auf zahlreiche Ereignisse reagie-

ren kann. Es ist empfehlenswert, solche Aktionen nicht unbedingt einem existierenden Movieclip zuzuweisen, der bereits eine eigene Aufgabe erfüllt. Stattdessen sollten Sie einen *leeren* Movieclip erstellen und auf die Bühne (oder in die gewünschte Zeitleiste) ziehen.

In Tabelle 6-7 sehen Sie zunächst eine Übersicht über die Ereignisse, die das Skript einer Movieclip-Instanz verarbeiten kann.

Tabelle 6-7: Übersicht der Ereignisse für Movieclip-Instanzaktionen

Ereignis	Beschreibung
`onClipEvent (load)`	erstes Auftreten der Instanz – praktisch für Initialisierungsarbeiten
`onClipEvent (unload)`	Verschwinden der Instanz aus der Zeitleiste – für Aufräumarbeiten am Ende einer Sequenz geeignet
`onClipEvent (enterFrame)`	Start jedes Frames – für regelmäßig wiederkehrende Aufgaben
`onClipEvent (mouseDown)`	Mausklick – nicht auf dem Movieclip, sondern irgendwo
`onClipEvent (mouseMove)`	Mausbewegung
`onClipEvent (mouseUp)`	Loslassen der Maustaste
`onClipEvent (keyDown)`	Drücken einer Taste auf der Tastatur
`onClipEvent (keyUp)`	Loslassen einer Taste auf der Tastatur
`onClipEvent (data)`	externe Daten werden geladen

Das nachfolgende Beispiel geht davon aus, dass sich auf der Bühne eine Movieclip-Instanz namens `test` befindet. Wenn Sie diesen Code wie beschrieben einem »Steuerclip« zuweisen, springt diese Instanz bei jedem Mausklick auf die Position des Mauszeigers und lässt sich mit der Maus ziehen, solange die Maustaste gedrückt bleibt:

```
onClipEvent (mouseDown) {
   startDrag (_root.test, true);
}

onClipEvent (mouseUp) {
   stopDrag();
}
```

Die Funktion `startDrag` (*`clipInstanz, zentrieren`*) sorgt dafür, dass die angegebene Instanz bis zum Aufruf von `stopDrag()` zusammen mit dem Mauszeiger bewegt wird. Das zweite Argument *`zentrieren`* bestimmt dabei, ob der Registrierpunkt des Movieclips auf die Mausposition springen soll (`true`) oder nicht (`false`).



Denken Sie daran, dass sich Movieclip-Instanzaktionen auf die Zeitleiste des Movieclips beziehen, dem sie zugeordnet sind. Um auf die Zeitleiste des Hauptfilms und deren Funktionen, Variablen und Methoden zugreifen zu können, müssen Sie diesen `_root` (oder den passenden relativen Bezug) voranstellen.

Praxisbeispiel: Navigation, interaktive Demo und Spiel

Die Anwendungsbeispiele in diesem Kapitel führen einen Schritt weiter: Statt der größtenteils passiven Animationen und Präsentationen der vorherigen Kapitel findet hier auch Interaktivität statt. Diese bildet eigentlich den wichtigsten Vorteil einer Flash-Anwendung gegenüber einer statischen HTML-basierten Website: Es *passiert* etwas, während die Seite bereits im Browser angezeigt wird. Konkret werden hier drei kleine Beispiele vorgestellt: Zuerst bekommt die Präsentation aus Kapitel 4 eine Steuerung. Anschließend wird eine interaktive Simulation vorgestellt, in der Sie verschiedene Gitarrenakkorde auswählen und anhören können. Zu guter Letzt erfahren Sie auch noch, wie ein kleines Spiel programmiert wird.

Navigation für die Produktschau

Der Ausgangspunkt für diese Anwendung ist im Prinzip mit dem Ergebnis des Praxisbeispiels aus Kapitel 4 identisch. Es wurden lediglich einige Schaltflächensymbole hinzugefügt. Sie finden die neue Datei unter *beispiele/kapitel6/katalog_start.fla* auf der CD zum Buch. Die Aufgabe für dieses Beispiel besteht darin, vier Schaltflächen hinzuzufügen, die (von links nach rechts) folgende Aufgaben erfüllen:

- Sprung zum vorigen Produkt
- Anhalten
- Weiterspielen
- Sprung zum nächsten Produkt

Wenn der Film angehalten ist, soll er auch nach einem Klick auf einen der Sprung-Buttons wieder stehen bleiben; falls er dagegen gerade läuft, wird er vom Sprungziel aus weiter abgespielt. Eine globale Variable namens `is_playing` trifft die zugehörige Entscheidung, ob `gotoAndPlay()` oder `gotoAndStop()` ausgeführt werden soll.

Öffnen Sie zunächst die Bibliothek, und wechseln Sie in den Ordner *buttons*; hier finden Sie die Schaltflächensymbole *back*, *pause*, *play* und *forward*. Designtechnisch sind diese bereits fertig. Das einzige, was ihnen noch fehlt, ist eine durchgehende Fläche im Frame *Aktiv*. Da einige von ihnen Lücken aufweisen, ist diese wichtig.

Doppelklicken Sie also in der Bibliothek auf jeden einzelnen der vier Buttons. Fügen Sie in deren Frame *Aktiv* mit **F7** ein leeres Schlüsselbild ein. Aktivieren Sie den Zwiebelschaleneffekt, damit der Inhalt des vorigen Frames durchscheint. Nun können Sie einfach ein Quadrat zeichnen, das den aktiven Bereich komplett abdeckt.

Kehren Sie zum Hauptfilm zurück. Markieren Sie die Schlüsselbilder im ersten Bild jeder Ebene – am einfachsten, indem Sie zuerst das Schlüsselbild in der obersten Ebene anklicken und dann mit gedrückter **Umschalt**-Taste das unterste. Ziehen Sie

dann sämtliche markierten Schlüsselbilder um zwei Frames nach rechts. Markieren Sie anschließend auf dieselbe Weise Bild 4 jeder Ebene (nun unmittelbar rechts von den verschobenen Schlüsselbildern). Drücken Sie zweimal **F5**, um die Dauer aller Ebenen wieder zu verlängern.

Legen Sie zwei neue Ebenen mit den Bezeichnungen *actions* beziehungsweise *labels* an. In Tabelle 6-8 finden Sie eine Übersicht darüber, auf welchen Frames dieser Ebenen Sie Schlüsselbilder einrichten müssen. Auf der Ebene *labels* werden die Schlüsselbilder mit den angegebenen Bildmarkierungen versehen, während auf *actions* die gezeigten Skripte erstellt werden müssen.

Tabelle 6-8: Erstellen Sie in den angegebenen Frames Schlüsselbilder mit Markierungen bzw. Skripten

Bild Nr.	Ebene	Markierung / Skript

2	*actions*	`// Initialisierung`
		`var is_playing = true;`
		`var prev_frame = "prod06";`
		`var next_frame = "prod01";`

3	*labels*	Markierung *start*
12	*labels*	Markierung *prod01*
12	*actions*	`prev_frame = "prod07";` `next_frame = "prod02";`
42	*labels*	Markierung *prod02*
42	*actions*	`prev_frame = "prod01";` `next_frame = "prod03";`
72	*labels*	Markierung *prod03*
72	*actions*	`prev_frame = "prod02";` `next_frame = "prod04";`
102	*labels*	Markierung *prod04*
102	*actions*	`prev_frame = "prod03";` `next_frame = "prod05";`
132	*labels*	Markierung *prod05*
132	*actions*	`prev_frame = "prod04";` `next_frame = "prod06";`
162	*labels*	Markierung *prod06*
162	*actions*	`prev_frame = "prod05";` `next_frame = "prod07";`
192	*labels*	Markierung *prod07*
192	*actions*	`prev_frame = "prod06";` `next_frame = "prod01";`
212	*actions*	`gotoAndPlay ("start");`

Erstellen Sie als Nächstes eine neue Ebene namens *buttons*. Ziehen Sie die vier Buttons in der oben angegebenen Reihenfolge auf die Bühne, und zwar mit etwas Abstand unterhalb des Logos. Weisen Sie dem Button *back* das folgende Skript zu:

```
on (release) {
    if (is_playing) {
       gotoAndPlay (prev_frame);
    } else {
        gotoAndStop (prev_frame);
    }
}
```

`prev_frame` ist eine globale Variable; sie enthält jeweils die Bezeichnung des Frames, auf dem das vorige Produkt angezeigt wird. Es gibt eine ähnliche Variable namens `next_frame`, die die Bezeichnung des Frames mit dem nachfolgenden Produkt enthält. Die passenden Werte werden jeweils automatisch beim Passieren der Bildskripte auf der Ebene *actions* festgelegt.

Danach erhält der Button *pause* folgendes Skript:

```
on (release) {
    is_playing = false;
    stop();
}
```

Der Schaltfläche *play* können Sie die folgenden ActionScript-Anweisungen zuweisen:

```
on (release) {
    is_playing = true;
    play();
}
```

Zu guter Letzt benötigt der Button *forward* noch ein Skript. Schreiben Sie die folgenden Anweisungen hinein:

```
on (release) {
    if (is_playing) {
       gotoAndPlay (next_frame);
    } else {
        gotoAndStop (next_frame);
    }
}
```

Eine interaktive Demo

Die nächste Beispielanwendung demonstriert die Steuerung von Movieclips: Sie können mit Hilfe von Schaltflächen einige Akkorde auswählen. Diese werden auf dem Griffbrett einer Gitarre angezeigt und gleichzeitig abgespielt. Die Gitarre ist ein Movieclip, der von außen angesteuert wird.

Öffnen Sie zunächst die Datei *demo_start.fla* aus dem Verzeichnis *beispiele/kapitel6* der CD-ROM. Die Bühne enthält wie üblich das Logo; in der Bibliothek finden Sie ein Symbol namens *griffbrett* sowie sechs importierte Sounds (die Akkorde E, A, H7, e, a und d).

Erzeugen Sie zuerst ein neues Symbol vom Typ Movieclip mit dem Namen *akkorde*. Ziehen Sie das Symbol *griffbrett* in dieses neue Symbol hinein, und platzieren Sie es mittig auf dem Registrierpunkt. Verlängern Sie die bisher einzige Ebene auf 35 Bilder, und benennen Sie sie in *griffbrett* um. Anschließend können Sie diese Ebene sperren.

Fügen Sie eine neue Ebene namens *griffe* ein. Sie erhält Schlüsselbilder in den Bildern 1, 6, 11, 16, 21, 26 und 31. Erstellen Sie eine weitere Ebene mit der Bezeichnung *labels*. Sie erhält Schlüsselbilder auf denselben Bildern. Diese werden von links nach rechts mit folgenden Bildmarkierungen versehen: *aus*, *E*, *A*, *H7*, *Em*, *Am* und *Dm*. Zeichnen Sie nun auf der Ebene *griffe*, angefangen mit dem Frame *E*, jeweils Punkte ein, die den Fingerpositionen der Akkorde entsprechen. Zusätzlich werden die Akkordbezeichnungen neben dem Griffbrett eingetragen (siehe Abbildung 6-11).

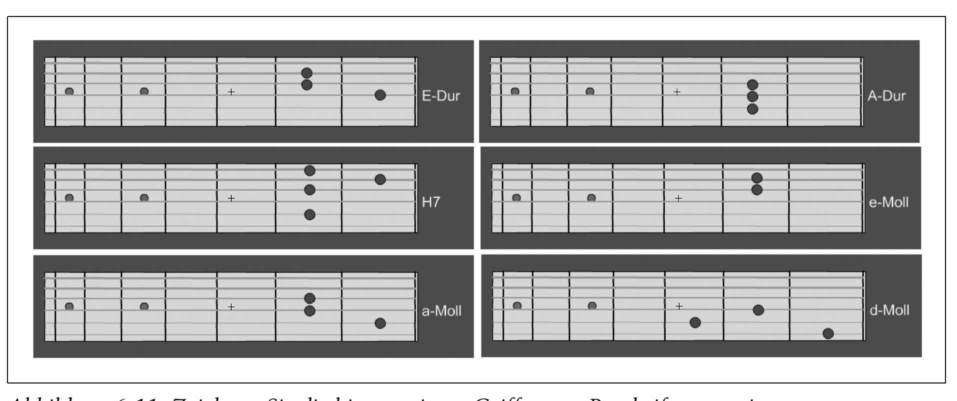


Abbildung 6-11: Zeichnen Sie die hier gezeigten Griffe samt Beschriftungen ein

Fügen Sie nun eine weitere Ebene mit dem Namen *sounds* ein. Sie benötigt Schlüsselbilder an denselben Positionen wie die anderen Ebenen. Weisen Sie den einzelnen Bildern die passenden Sounds zu, und zwar jeweils mit den Einstellungen *Sync: Anfang* und *Endlosschleife*. Als Letztes wird noch eine Ebene namens *actions* eingefügt. Fügen Sie auch auf dieser die üblichen Schlüsselbilder ein, und weisen Sie jedem von ihnen die folgende Anweisung zu:

```
stop();
```

Die Zeitleiste des Symbols *akkorde* muss nun so aussehen wie in Abbildung 6-12.

Wechseln Sie wieder zurück zum Hauptfilm. Ziehen Sie eine Instanz des Symbols *akkorde* auf die Bühne, und weisen Sie ihr den Instanznamen *akk* zu.

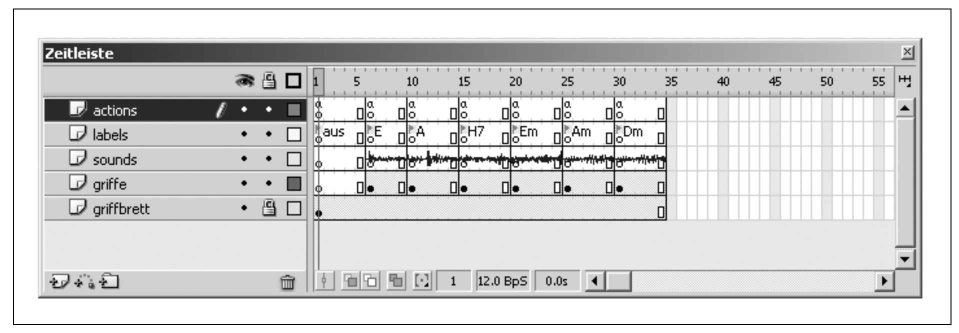


Abbildung 6-12: Die Zeitleiste des Symbols »akkorde«

Ziehen Sie nun die sieben Schaltflächen aus dem Bibliotheksordner *buttons* auf die Bühne, so dass sie nebeneinander unter dem Griffbrett liegen. Weisen Sie dem Button *aus* folgendes Skript zu:

```
on (release) {
   stopAllSounds(); //Wichtig!
   akk.gotoAndStop ("aus");
}
```

Verfahren Sie mit den sechs anderen Buttons ebenso. Die Sprungziele in der Instanz `akk` sind natürlich mit den Beschriftungen der Schaltflächen selbst identisch.

Ein Bildpaare-Suchspiel

Spiele sind seit spätestens Flash 4 eine der beliebtesten Anwendungen des Programms. In diesem Abschnitt wird deshalb die Flash-Version eines beliebten und bekannten Bildpaare-Suchspiels vorgestellt.[5]

Öffnen Sie die Datei *spiel_start.fla* aus dem Verzeichnis *beispiele/kapitel6* der Buch-CD. Da es in diesem Beispiel vor allem um ActionScript geht, sind alle anderen Arbeiten – beispielsweise die Erstellung von Symbolen und deren Platzierung auf der Bühne – bereits erledigt.

Das Konzept

Das Kernelement der Anwendung ist ein Movieclip-Symbol namens *karte*. Es enthält das Deckblatt und die sechs unterschiedlichen Kartenbilder des Spiels in aufeinander folgenden Frames; das achte Frame ist ein leeres Schlüsselbild, das angezeigt wird, wenn das entsprechende Kartenpaar abgeräumt wurde. Über den ersten sieben Frames befindet sich eine transparente Schaltfläche, die dem »Auf-

5 Der Name, unter dem das Spiel bekannt ist (die englische Bezeichnung für »Arbeitsspeicher«), darf aus Copyright-Gründen nicht genannt werden – dabei wäre es hervorragende Gratiswerbung für das Original ...

decken« der Karte dient. Sie ruft die im Hauptfilm definierte Funktion `klick()` auf, wobei sie mittels `this` einen Verweis auf die aktuelle Instanz von *karte* übermittelt, damit die Funktion weiß, welche Karte angeklickt wurde.

Auf der Bühne befinden sich insgesamt zwölf Instanzen des Symbols *karte*; sie tragen die Instanznamen *karte0* bis *karte11*.

Das Hauptskript muss zunächst einmal dafür sorgen, dass die sechs Kartenpaare zufällig auf diese zwölf Positionen verteilt werden. Zu diesem Zweck werden zwei Arrays definiert: Eines enthält zwölf unterschiedliche Bezeichnungen (*1a*, *1b*, *2a*, *2b* und so weiter), damit sichergestellt ist, dass jedes Kartenpaar nur einmal vorhanden ist. Im zweiten Array befinden sich – jeweils paarweise – die sechs Zahlen von 2 bis 7 (also *2*, *2*, *3*, *3* und so weiter). Sie entsprechen den Nummern der Frames, auf denen im Symbol *karte* die einzelnen Kartenbilder liegen.

Der »Mischprozess« funktioniert nun folgendermaßen: Per Zufallsgenerator wird ein Wert von 0 bis 11 ausgewählt und als Index auf die beiden Arrays angewendet. Das Ganze funktioniert genauso wie das weiter oben vorgestellte Lotto-Beispiel.

Als Nächstes sollten Sie verstehen, was die Funktion `klick()` tut, die beim Anklicken einer Kartenposition aufgerufen wird. Sie arbeitet mit einer Variablen namens `offene`, um zu unterscheiden, ob gerade die erste oder die zweite Karte eines Paares geöffnet wird. Wenn die Variable den Wert 1 hat, eine Karte also bereits aufgedeckt ist, wird zuerst einmal geprüft, ob der Klick zum zweiten Mal auf dieselbe Karte erfolgt. Ist dies der Fall, dann wird die Funktion mittels `return` verlassen – der Klick wird nicht gewertet.

Sind dagegen schon zwei Karten geöffnet, dann werden diese *sofort* geschlossen, bevor die aktuell angeklickte Instanz als erste Karte eines neuen Paares gewertet wird. Normalerweise werden zwei geöffnete Karten erst nach einer Sekunde automatisch geschlossen beziehungsweise – wenn sie identisch sind – abgeräumt.

Nach dem Öffnen der jeweils zweiten Karte wird eine zusätzliche Prüfung durchgeführt: Sind beide Karten identisch (gleiche Frame-Nummer), dann wird die spezielle Variable `gleich` auf `true` gesetzt; zudem wird der Paare-Zähler `paare` um 1 erhöht.

Für das automatische Schließen ist ein Steuerclip zuständig: Über den Event-Handler `onClipEvent(enterFrame)` überprüft er regelmäßig (einmal pro Frame), ob zwei Karten geöffnet sind und – wenn dies der Fall ist – ob die durch die Variable `warten` definierte Wartezeit bereits abgelaufen ist. Die eigentliche Arbeit des Schließens beziehungsweise Abräumens erledigt allerdings die Funktion `schliessen()`. Diese wird übrigens auch dann aktiviert, wenn der Benutzer innerhalb der Wartezeit eine weitere Karte anklickt.

Die Skripte

Ihre Aufgabe für dieses Praxisbeispiel besteht im Verfassen mehrerer ActionScript-Blöcke. Wechseln Sie zunächst in das Symbol *karte*. Weisen Sie dem Schlüsselbild der Ebene *actions* die folgende Anweisung zu:

```
stop();
```

Dies sorgt natürlich dafür, dass die Karten-Instanzen das Deckblatt anzeigen, anstatt automatisch in einer Schleife abgespielt zu werden. Als Nächstes benötigt auch die transparente Schaltfläche auf der Ebene *button* ein Skript:

```
on (release) {
     _root.klick (this);
}
```

Wechseln Sie nun in den Hauptfilm. Hier werden auf der Ebene *actions* mehrere Bildskripte erstellt. Wählen Sie zunächst Bild 2 aus, und tragen Sie das folgende umfangreiche Skript in die Aktionen-Palette ein:

```
// Kartenvorrat und zugehörige Bilder im Symbol karte
var kartenstapel = new Array ("1a", "1b", "2a", "2b", "3a", "3b", "4a", "4b", "5a",
 "5b", "6a", "6b");
var kartenframes = new Array (2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7);
// Zufällige Verteilung der Karten
// Die 12 Kartenpositionen
var karten = new Array ();
for (var i = 0; i < 12; i++) {
   var k = Math.floor (Math.random () * kartenstapel.length);
   // Karte und Frame wählen
   karte = kartenstapel[k];
   bild = kartenframes[k];
   // Entsprechende Elemente aus den Arrays entfernen
   kartenstapel.splice (k, 1);
   kartenframes.splice (k, 1);
   // Karte speichern
   karten[i] = karte;
   // Der aktuellen Instanz ihre Frame-Nummer mitteilen
   var k_ref = eval ("karte" + i);
   k_ref.bildNr = bild;
}
// Statusvariablen
// bisher kein Paar gefunden
var paare = 0;
// Anzahl der offenen Karten
var offene = 0;
// beide Karten gleich?
var gleich = false;
// Wartezeit bis zum autom. Schließen
var warten = 0;
// Anzahl der Versuche
var versuche = 0;
// Feldnummern der geöffneten Karten (zum Schließen)
var pos = new Array ();
```

```
function klick (karteMC)
{
   // Nichts tun, wenn zweite Karte wie erste
   if (offene == 1 && pos[1] == karteMC) {
      return;
   }
   // Sind schon zwei Karten offen?
   if (offene == 2) {
      // beide schließen
        schliessen();
   }
   // Karte öffnen
   karteMC.gotoAndStop (karteMC.bildNr);
   offene++;
   // Karte merken
   pos[offene] = karteMC;
   // Ist dies die 2. Karte?
   if (offene == 2) {
      // sind die Karten gleich?
      if (pos[1].bildNr == pos[2].bildNr) {
         gleich = true;
         paare++;
      } else {
         gleich = false;
      }
   }
}

function schliessen ()
{
   offene = 0;
   warten = 0;
   versuche++;
   // War ein Paar offen?
   if (gleich) {
      pos[1].gotoAndStop (8);
      pos[2].gotoAndStop (8);
      if (paare == 6) {
         gotoAndStop ("win");
      }
   } else {
      pos[1].gotoAndStop (1);
      pos[2].gotoAndStop (1);
   }
}
```

Genauere Erläuterungen zur Funktion dieses Codes haben Sie bereits im vorigen Unterabschnitt erhalten.

Das Skript in Frame 3 ist im Gegensatz dazu sehr kurz:

```
stop();
```

Selbstverständlich ist es auch ohne Weiteres möglich, diese Anweisung im Skript von Frame 2 unterzubringen. Allerdings ist es übersichtlicher, Initialisierungscode und Funktionsdefinitionen von einfachen Navigationsanweisungen zu trennen.

In Frame 20 (unter der Markierung *win*) können Sie folgendes Skript einfügen, das die Anzahl der benötigten Versuche und einen passenden Kommentar ausgibt:

```
ausgabe = "Sie haben " + versuche + " Versuche benötigt.\n";
if (versuche < 6) {
   ausgabe += "Haben Sie das Spiel bestochen oder gehackt?";
} else if (versuche == 6) {
   ausgabe += "Gepfuscht, geraten oder Glück gehabt?!";
} else if (versuche < 12) {
   ausgabe += "Sehr gut gemacht!";
} else if (versuche < 18) {
   ausgabe += "Recht ordentliche Leistung!";
} else {
   ausgabe += "Sie sollten noch ein wenig üben!";
}
stop();
```

Im gleichen Frame finden Sie auf der Ebene *karten* die Schaltfläche *Neues Spiel*. Weisen Sie ihr folgende Anweisungen zu:

```
on (release) {
   gotoAndStop ("start");
}
```

Wechseln Sie nun wieder in das Start-Schlüsselbild der Ebene *karten*. Ziehen Sie das Symbol *steuerclip* auf (oder besser neben) die Bühne. Klicken Sie es an, und versehen Sie es mit dem folgenden Code:

```
onClipEvent (enterFrame) {
   // muss gerade gewartet werden?
   if (_root.offene == 2) {
      _root.warten++;
      // eine Sekunde vorbei?
      if (_root.warten >= 12) {
         _root.schliessen();
      }
   }
}
```

In diesem Kapitel:

- Objektorientierte Programmierung mit ActionScript
- Klassen entwerfen und programmieren
- Praxisbeispiel: Eine Komfort-Datumsklasse

KAPITEL 7

ActionScript 2.0-Klassen

Die Ordnung und Verknüpfung der Ideen ist dieselbe
wie die Ordnung und Verknüpfung der Dinge.
– Baruch de Spinoza

Die wichtigste Neuerung der ActionScript-Version 2.0 ist die Objektorientierung mit echten Klassen. Konzept und Umsetzung wurden im Großen und Ganzen von der Programmiersprache Java inspiriert. In diesem Kapitel erfahren Sie zunächst anhand eines praktischen Beispiels, wie Objektorientierung funktioniert und wofür sie nützlich ist – vor allem im direkten Vergleich mit dem herkömmlichen Programmieransatz. Anschließend wird erläutert, wie Sie ActionScript-Klassen schreiben und einsetzen können. Zu guter Letzt gibt es wie immer ein ausführliches Praxisbeispiel.



ActionScript war im Grunde von Anfang an objektorientiert, da die eingebauten Elemente von Flash über Eigenschaften und Methoden gesteuert werden. Diese »passive« Objektorientierung haben Sie bereits im vorigen Kapitel kennen gelernt, etwa bei der Steuerung von Movieclips. Konstrukte wie `meinClip.gotoAndPlay (17)` oder `meinClip._x` sind typische OO-Elemente; es handelt sich um einen Methodenaufruf beziehungsweise um den Zugriff auf eine Eigenschaft. Auch eigene Klassen und Objekte konnte man entwerfen, allerdings über eine wenig eingängige Syntax mit Funktionen. Die entscheidende Neuerung in ActionScript 2.0 besteht daher in der neuen, aufgeräumten Klassenschreibweise, die zugleich einige Erweiterungen der Objektorientierungsfähigkeiten mit sich bringt.

Bitte beachten Sie, dass es in diesem Kapitel weniger grafisch zugeht als bisher in diesem Buch. Auch das Praxisbeispiel ist zwar überaus nützlich, erzeugt aber keine sichtbare Ausgabe. Dafür können Sie die dort vorgestellte Klasse nicht nur für das FlashRock-MusicShop-Beispiel nutzen, sondern auch für Ihre eigenen Projekte.

Objektorientierte Programmierung mit ActionScript

Das Konzept der *Objektorientierten Programmierung* (OOP) wurde bereits in den 70er Jahren des vorigen Jahrhunderts ersonnen. Es entstand – wie so viele Errungenschaften der Computertechnik – im Forschungslabor XEROX PARC in Palo Alto, Kalifornien. Dort wurde schon vor 30 Jahren eine Arbeitsumgebung entwickelt, die erst seit den 90er Jahren allgemein verbreitet ist: vernetzte Arbeitsplatzrechner mit grafischer Benutzeroberfläche inklusive Maus und WYSIWYG-Dokumentenbearbeitung.

Zur Implementierung dieses Systems entwarfen die Forscher eine Programmiersprache namens *Smalltalk* und zugleich eine völlig neue Arbeitsweise für die Softwareentwicklung: Die objektorientierte Programmierung wird ergänzt durch die Verfahren der objektorientierten Analyse (OOA) zur Beschreibung der gewünschten Ergebnisse und des objektorientierten Designs (OOD) für den Entwurf der Softwarearchitektur.[1]

Smalltalk ist leicht zu lernen, besitzt eine elegante und schlanke Syntax und ist perfekt in eine Arbeitsumgebung eingepasst, die sich auf Knopfdruck beliebig umprogrammieren lässt.[2] Dennoch wurde der objektorientierte Ansatz erst viel später populär, nämlich durch die Erweiterung der klassischen Sprache C zum objektorientierten C++. Aufgrund dieser Erfahrungen wurde 1995 die bisher populärste OOP-Sprache Java entwickelt; im Jahr 2000 folgte eine ähnliche Sprache von Microsoft namens C#.

Das Grundkonzept der objektorientierten Programmiersprachen heißt *Kapselung*. Dabei werden die gespeicherten Datenstrukturen und die Funktionen zur Manipulation derselben zusammengefasst – das Ergebnis sind die *Objekte*, die genau wie Gegenstände in der realen Welt »selbstständig agieren« können. Daher eignet sich die OOP hervorragend zur Nachbildung von Gegenständen, Arbeits- oder Geschäftsabläufen aus der Realität. Auch zur Erstellung grafischer Benutzeroberflächen mit ihren Fenstern, Menüs und Schaltflächen ist die Objektorientierung ideal geeignet.

Schwächen der klassischen Programmierung

In der herkömmlichen – der so genannten imperativen oder auch prozeduralen – Programmierung werden die Datenstrukturen und die Funktionen getrennt voneinander entworfen. Betrachten Sie als Beispiel einige Aspekte eines Autos (*der* Klassiker zur Erläuterung der OO-Vorteile): Es tankt, maximal bis zur Höchstkapazität

---

1 Die allererste objektorientierte Programmiersprache war Smalltalk allerdings nicht; diese Ehre gebührt Simula (siehe etwa *http://de.wikipedia.org/wiki/Simula*).

2 Wer es ausprobieren möchte, kann die Website *http://www.squeak.org* besuchen und sich die vollständige Smalltalk-Umgebung *Squeak* für viele verschiedene Plattformen herunterladen.

seines Tanks, und es fährt, wobei es (idealisiert) stets eine feste Anzahl von Litern Kraftstoff pro 100 km verbraucht. Für das »imperative Auto« werden diese Daten in Variablen gespeichert, und Funktionen greifen darauf zu, um diese Daten zu ändern. Den folgenden Code, der das demonstriert, können Sie dem ersten Bild eines leeren Films zuweisen:

```
// Eigenschaften des Autos
var maxTankFuellung = 50;  // Kapazität des Tanks 50 l
var verbrauch = 10;        // Verbrauch 10 l/100 km
// idealer Neuwagen: voller Tank, 0 km
var tankFuellung = maxTankFuellung;
var kmStand = 0;

// Funktionen: Fahren und Tanken; greifen auf globale Variablen zu
function tanken (l)
{
   // Nur tanken, falls die gewünschte Menge in den Tank passt
   if (tankFuellung + l <= maxTankFuellung) {
      tankFuellung += l;
   }
}

function fahren (km)
{
   // Nur fahren, falls der Sprit reicht
   if (km * verbrauch / 100 <= tankFuellung) {
      kmStand += km;
      tankFuellung -= km * verbrauch / 100;
   }
}

// Funktion auskunft(): Ausgabe vereinfachen
function auskunft()
{
   trace ("Das Auto hat " + tankFuellung +
        " l im Tank und den Kilometerstand " + kmStand);
}

// Anfangszustand
auskunft();
// 250 km fahren
fahren (250);
auskunft();
// 10 l tanken
tanken (10);
auskunft();
```

Die ActionScript-Anweisungen in diesem Beispiel müssten Sie verstehen, wenn Sie das vorige Kapitel durchgearbeitet haben.

Führen Sie das Beispiel mit Hilfe von *Steuerung* → *Film testen* (**STRG + ENTER**) aus. Es erzeugt durch Aufruf der Funktion `auskunft()` folgende drei Zeilen im Ausgabefenster:

```
Das Auto hat 50 l im Tank und den Kilometerstand 0
Das Auto hat 25 l im Tank und den Kilometerstand 250
Das Auto hat 35 l im Tank und den Kilometerstand 250
```

Solange Sie nur ein »Auto« verwalten möchten, bleibt die imperative Lösung noch einigermaßen übersichtlich. Aber stellen Sie sich vor, auf diese Weise sollten die Eigenschaften eines ganzen Fuhrparks bearbeitet werden. Sie müssten die Daten aller Fahrzeuge beispielsweise in Arrays verwahren und den Funktionen jeweils den Index des aktuellen Autos übergeben. Hier nur ein abgespecktes Beispiel für drei Autos mit `tankFuellung`, `maxTankFuellung` und der Funktion `tanken()`:

```
// Eigenschaften der Autos
var maxTankFuellung = [30, 40, 50];  // Kapazitäten der Tanks
var tankFuellung = [10, 10, 10];     // Anfangs-Tankfüllungen
var kmStand = 0;
// Hier fehlen weitere Eigenschaften

function tanken (autoID, l)
{
   // Nur tanken, falls die gewünschte Menge in den Tank passt
   if (tankFuellung[autoID] + l <= maxTankFuellung[autoID]) {
      tankFuellung[autoID] += l;
   }
}
// Hier fehlt die Funktion fahren()

function auskunft(autoID)
{
   trace ("Das Auto " + autoID + " hat " +
          tankFuellung[autoID] +  " l im Tank.");
}

// Schleife zum Betanken aller Autos
for (i = 0; i < 3; i++) {
   // Anfangszustand
   auskunft(i);
   // Je 10 l tanken
   tanken (i, 10);
   // Info
   auskunft(i);
}
```

Die Ausgabe des Beispiels sieht so aus:

```
Das Auto 0 hat 10 l im Tank.
Das Auto 0 hat 20 l im Tank.
Das Auto 1 hat 10 l im Tank.
Das Auto 1 hat 20 l im Tank.
Das Auto 2 hat 10 l im Tank.
Das Auto 2 hat 20 l im Tank.
```

Diese Variante ist wirklich nicht mehr übersichtlich. Das ist aber gar nicht das größte Problem – noch schlimmer ist die Tatsache, dass sich die Werte der globalen Variablen an beliebiger Stelle ändern lassen. Schauen Sie sich das folgende, auf die Funktionen aus dem Einzelauto-Beispiel bezogene, Szenario an:

```
// Anfangszustand: km-Stand 0, Tankfüllung 50 l
//                 max. Tankfüllung 50, Verbrauch 10 l/100 km

auskunft();
// 250 km fahren
fahren(250);
// Tacho "zurückdrehen"
kmStand = 0;
// Info
auskunft();
```

Die Ausgabe zeigt, dass etwas nicht stimmen kann:

```
Das Auto hat 50 l im Tank und den Kilometerstand 0
Das Auto hat 25 l im Tank und den Kilometerstand 0
```

Natürlich machen Programmierer solche Fehler in der Regel nicht mit Absicht. Aber solange Datenstrukturen und Funktionen keine Einheit bilden, kann es leicht dazu kommen. Es ist schwierig und zeitaufwändig, derartige Probleme im Nachhinein aufzuspüren.

Der praktische Nutzen der Objektorientierung

Um die oben beschriebenen Probleme in den Griff zu bekommen, können die Stärken des objektorientierten Ansatzes zur Geltung kommen: Durch die genannte Kapselung von Datenstrukturen und Funktionen zur Einheit des Objekts ist gewährleistet, dass die Daten nur noch über zulässige Schnittstellen geändert werden. Das objektorientierte Auto kann also »selbst« `fahren()` und `tanken()`.

In der Praxis funktioniert objektorientierte Programmierung in ActionScript und in den meisten anderen modernen Sprachen so:

1. Mit Hilfe der Konstruktion `class` *`Klassenname`* `{...}` wird eine *Klasse* definiert. Es handelt sich um einen Datentyp beziehungsweise eine Vorlage zum Erstellen von Objekten einer bestimmten Sorte. Die Klasse kann von einer anderen Klasse abgeleitet sein (Vererbung); in diesem Fall müssen nur Unterschiede und Ergänzungen neu programmiert werden.
2. Die Datenstruktur wird in Form so genannter *Eigenschaften* oder *Attribute* innerhalb der Klasse gespeichert. Es handelt sich um Variablen, auf die in der Regel kein direkter Zugriff von außen besteht. In Einzelfällen können Sie allerdings auch öffentliche Eigenschaften definieren – denken Sie nur an die im vorigen Kapitel besprochenen `MovieClip`-Eigenschaften wie `_alpha` oder `_rotation`.

3. Wenn mit Hilfe des bereits intuitiv verwendeten Operators `new()` ein neues Objekt einer Klasse erstellt wird, wird eine spezielle Funktion aufgerufen, nämlich der *Konstruktor*. Um einen eigenen Konstruktor zu definieren, müssen Sie innerhalb der Klasse eine Funktion erstellen, die denselben Namen trägt wie die Klasse selbst. Der Konstruktor wird für Initialisierungsaufgaben eingesetzt – beispielsweise, um den Eigenschaften ihre Anfangswerte zuzuweisen.
4. Die öffentliche Funktionalität der Klasse wird durch ihre *Methoden* bereitgestellt. Dies sind im Grunde beliebige Funktionen, die später über die aus der Klasse erzeugten Objekte aufgerufen werden.
5. Nachdem die Klasse erstellt wurde, können Sie beliebig viele Objekte – so genannte *Instanzen* der Klasse – erstellen; die grundlegende Syntax sieht so aus:
   ```
   var instanzName:KlassenName =
                   new KlassenName ([Argument [, Argument ...]]);
   ```
6. Die Instanz kann nun über die in der Klasse definierten Methoden manipuliert werden. Die Schreibweise für die Methodenaufrufe lautet:
   ```
   instanzName.methodenName ([Argument [, Argument ...]]);
   ```
7. Wenn Sie möchten, können Sie von der soeben entwickelten Klasse wieder andere Klassen ableiten.

Eine Klassendefinition muss in einer separaten ActionScript-Datei mit der Endung *.as* gespeichert werden. Der Dateiname vor der Erweiterung muss *exakt* dem Klassennamen entsprechen, und zwar einschließlich Groß- und Kleinschreibung, obwohl Windows diese nie und Mac OS X sie nur auf UFS-Partitionen unterstützt. Der ActionScript-Compiler in der SWF-Exportfunktion findet solche Klassendateien automatisch, wenn Sie sie im gleichen Verzeichnis speichern wie die darauf zugreifenden Flash-Filme.

Um innerhalb von Flash eine ActionScript-Datei zu erstellen, wählen Sie *Datei → Neu* und dann den Typ *ActionScript-Datei* von der Registerkarte *Allgemein*. Das Hauptfenster dieser Datei entspricht im Wesentlichen dem im vorigen Kapitel beschriebenen Bedienfeld *Aktionen*, allerdings stehen die meisten Optionen diesmal im Menü *Ansicht* zur Verfügung.

Hier zunächst eine einfache Fassung der Klasse `Auto`. Im nächsten Abschnitt wird sie im Rahmen der Beschreibung diverser Themen noch einige Male erweitert. Erstellen Sie wie beschrieben eine neue ActionScript-Datei, und geben Sie den folgenden Code ein; weiter unten erhalten Sie eine genaue Erläuterung.

```
class Auto
{
   // Eigenschaften
   private var kmStand:Number;
   private var tankFuellung:Number;
   private var maxTankFuellung:Number;
   private var verbrauch:Number;
```

```
    // Konstruktor
    public function Auto (m:Number, v:Number)
    {
       this.maxTankFuellung = m;
       this.verbrauch = v;
       // Zu Anfang: km-Stand 0, Tank voll
       this.kmStand = 0;
       this.tankFuellung = this.maxTankFuellung;
    }

    // Methoden
    public function fahren (km:Number):Void
    {
       // Nur fahren, falls das Benzin reicht
       if (km * this.verbrauch / 100 <= this.tankFuellung) {
          kmStand += km;
          tankFuellung -= km * this.verbrauch / 100;
       }
    }

    public function tanken (l:Number):Void
    {
       // Nur tanken, falls die Kapazität ausreicht
       if (this.tankFuellung + l <= this.maxTankFuellung) {
          tankFuellung += l;
       }
    }

    public function getKmStand():Number
    {
       return this.kmStand;
    }

    public function getTankFuellung():Number
    {
       return this.tankFuellung;
    }

    public function getMaxTankFuellung():Number
    {
       return this.maxTankFuellung;
    }

    public function getVerbrauch():Number
    {
       return this.verbrauch;
    }
}
```

Die Klasse selbst ist *keine* vollständige Flash-Anwendung, so dass die Funktion *Film testen* keine Wirkung hat. Sie können allerdings *Werkzeuge → Syntax überprüfen* oder die Häkchen-Schaltfläche in der Symbolleiste verwenden, um die formale Korrektheit des Codes zu überprüfen.

Bevor es mit Erklärungen weitergeht, soll die Klasse eingesetzt werden, um »objekt-orientierte Autos« zu erstellen und sie `fahren()` und `tanken()` zu lassen. Speichern Sie die Klassendatei dazu unter dem Namen *Auto.as*, und erstellen Sie dann einen neuen Flash-Film. Geben Sie in der Aktionen-Palette das folgende Skript für das erste Bild ein:

```
function auskunft (a:Auto):Void
{
   trace ("Tankfüllung: " + a.getTankFuellung() + ", " +
          "Kilometerstand: " + a.getKmStand());
}

var auto:Auto = new Auto (50, 10);
trace ("Anfangszustand:");
auskunft(auto);
auto.fahren (250);
trace ("Nach 250 km Fahrt:");
auskunft(auto);
auto.tanken (10);
trace ("Nach 10 l Tanken:");
auskunft(auto);
```



Ein Flash-Film, der auf selbst definierte Klassen zurückgreift, *muss* gespeichert werden, bevor Sie *Film testen* verwenden – standardmäßig im gleichen Verzeichnis wie die *.as*-Dateien mit den Klassendefinitionen. Wenn Sie einen ungespeicherten Film testen, wird die SWF-Datei nämlich in einem Temporär-Verzeichnis erstellt, so dass die Klassen nicht gefunden werden. Darüber erhalten Sie noch nicht einmal eine Fehlermeldung, sondern es kommt kommentarlos zu Fehlfunktionen.

Die Ausgabe dieses Beispiels sieht folgendermaßen aus:

```
Anfangszustand:
Tankfüllung: 50, Kilometerstand: 0
Nach 250 km Fahrt:
Tankfüllung: 25, Kilometerstand: 250
Nach 10 l Tanken:
Tankfüllung: 35, Kilometerstand: 250
```

Die Klasse selbst wird von folgendem Konstrukt umschlossen:

```
class Auto
{
   // Klassencode
}
```

Die typische Reihenfolge innerhalb des Blocks mit den Klassenelementen ist:

1. Deklaration der Eigenschaftsvariablen
2. Definition des Konstruktors
3. Definition der Methoden

Eigenschaften sind im Grunde völlig normale Variablen, wie Sie sie bereits im vorigen Kapitel kennen gelernt haben. Wenn Sie objektorientiert in ActionScript 2.0 programmieren, sollten Sie bevorzugt von festen Datentypen Gebrauch machen – die Eigenschaften des vorliegenden Beispiels sind etwa alle numerisch, so dass sie den Typ `Number` haben:

```
private var kmStand:Number;
private var tankFuellung:Number;
private var maxTankFuellung:Number;
private var verbrauch:Number;
```

Das Schlüsselwort `private` bestimmt die Geheimhaltungsstufe der Eigenschaften: Sie sind nur innerhalb des Klassenblocks selbst sichtbar; ein Zugriff von außen ist nicht möglich. Betrachten Sie das folgende ungültige Beispiel:

```
var a:Auto = new Auto(35, 6);
trace (a.maxTankFuellung);
```

Das Ergebnis dieses Versuchs ist eine Fehlermeldung wie diese:

```
**Fehler** Szene=Szene 1, Ebene=Ebene 1, Bild=1:Zeile 4:
 Keine Eigenschaft mit dem Namen 'maxTankFuellung' vorhanden.
     trace (a.maxTankFuellung);
```

Die Eigenschaft ist nach außen, in Bezug auf eine Instanz der Klasse `Auto`, gar nicht bekannt! Eines der Ziele der Kapselung – den Zugriff auf die Datenstruktur nur noch durch offizielle, kontrollierte Schnittstellen zu erlauben – wurde offensichtlich erreicht.

Der Konstruktor und die Methoden besitzen dagegen die Geheimhaltungsstufe `public`, da sie von außen verfügbar sein sollen. Genau sie sind eben die definierten Schnittstellen zur Datenstruktur.

Der Konstruktor der Klasse `Auto` nimmt zwei Parameter entgegen – die maximale Tankfüllung und den Verbrauch in l pro 100 km. Anschließend weist er allen vier Eigenschaften ihre Anfangswerte zu:

```
public function Auto (m:Number, v:Number)
{
   this.maxTankFuellung = m;
   this.verbrauch = v;
   // Zu Anfang: km-Stand 0, Tank voll
   this.kmStand = 0;
   this.tankFuellung = this.maxTankFuellung;
}
```

Wie Sie sehen, werden der maximalen Tankfüllung und dem Verbrauch die übergebenen Werte zugewiesen. Der Kilometerstand wird auf 0, die aktuelle Tankfüllung auf das erlaubte Maximum gesetzt. Ein Konstruktor besitzt keine Datentypangabe, da der Rückgabewert eines `new()`-Aufrufs eine Instanz der Klasse selbst ist.

Das Schlüsselwort `this` haben Sie bereits im Zusammenhang mit Movieclips kennen gelernt. Auch in einer Klassendefinition bezieht es sich nicht etwa auf die Klasse selbst (die Vorlage), sondern auf die aktuelle *Instanz*, für die der Konstruktor (mittels `new()`) oder die jeweilige Methode gerade ausgeführt wird. Im vorliegenden Fall dient es also dem Bezug auf die Eigenschaftsvariablen der soeben erzeugten Instanz.



Es ist absolut legal, das Schlüsselwort `this` wegzulassen, solange es innerhalb des Konstruktors oder der Methode keine gleichnamige lokale Variable gibt. Allerdings sollten Sie es sich für den Zugriff auf die Eigenschaften *grundsätzlich* angewöhnen, `this` zu verwenden. So wird auch ohne Blättern in einer langen Klassendefinition immer sofort klar, dass eine Eigenschaft und keine lokale Variable gemeint ist. Wenn Sie eine Klasse erst einige Monate nach ihrer Erstellung wieder bearbeiten, wird Ihnen dies die Arbeit enorm erleichtern.[3]

Nach dem Konstruktor folgen die Methoden. Die beiden ersten erledigen die eigentlichen Aufgaben des Autos: eine bestimmte Anzahl von Kilometern fahren, falls der Sprit reicht, beziehungsweise die angegebene Menge Benzin tanken, wenn es in den Tank passt. Im Prinzip funktioniert beides genau wie beim »imperativen Auto« und dürfte verständlich sein. Betrachten Sie als Beispiel die Methode `tanken()`:

```
public function tanken (l:Number):Void
{
   // Nur tanken, falls die Kapazität ausreicht
   if (this.tankFuellung + l <= this.maxTankFuellung) {
      tankFuellung += l;
   }
}
```

Der Datentyp von `fahren()` und `tanken()` ist `Void` – es handelt sich um eine spezielle Angabe für Funktionen, die *keinen* Wert zurückgeben. Demzufolge kann dieser Typ natürlich nicht für Variablen verwendet werden.

Die restlichen Methoden sind so genannte *Accessor-Methoden* – sie erlauben den indirekten Zugriff auf Eigenschaften, die eigentlich privat sind. Dies scheint dem Ansatz der Kapselung zu widersprechen. In Wirklichkeit können Sie sich aber im Einzelfall aussuchen, für welche Eigenschaften Sie Accessor bereitstellen und ob diese mit Beschränkungen versehen werden sollen. Typische Accessor-Methoden lassen sich in *Getter-* und *Setter-Methoden* unterteilen, heißen entsprechend `get`*`Eigenschaft`*`()` oder `set`*`Eigenschaft`*`()` und dienen dem Auslesen beziehungsweise Ändern der jeweiligen Eigenschaft.

---

3 Dies ist aber nur meine Erfahrung – manche Programmierer fühlen sich auch ohne `this` wohl.

Die Klasse Auto definiert nur Getter- und keine Setter-Methoden, da die Eigenschaften nur gelesen, aber nicht direkt geändert werden sollen. Für die Manipulation sind fahren() und tanken() zuständig, die nur zwei der vier Eigenschaften überhaupt ändern und vor den Änderungen Plausibilitätstests durchführen. Der Datentyp einer Getter-Methode entspricht dem Typ der jeweiligen Eigenschaft; die einzige Anweisung ist return this.*Eigenschaft*. Hier zum Beispiel die Methode getKmStand:

```
public function getKmStand():Number
{
   return this.kmStand;
}
```

Setter-Methoden lernen Sie weiter unten in diesem Kapitel kennen.

Der Flash-Film zum Testen der Klasse definiert zunächst eine etwas modifizierte Funktion auskunft():

```
function auskunft (a:Auto):Void
{
   trace ("Tankfüllung: " + a.getTankFuellung() + ", " +
          "Kilometerstand: " + a.getKmStand());
}
```

Ihr Übergabeparameter ist diesmal eine Auto-Instanz. Beachten Sie, dass bei einer solchen Übergabe kein neues Objekt erstellt wird; es handelt sich vielmehr um eine *Referenz* auf die übergebene Instanz. Sollten in der Funktion Methoden der Instanz aufgerufen oder ihre öffentlichen Eigenschaften geändert werden, hat dies Auswirkungen auf das Übergabeobjekt. In auskunft() werden allerdings nur Getter-Methoden verwendet, so dass sich keine Änderungen der Instanz ergeben.



Wenn Sie einfache, eingebaute Datentypen wie Number, String oder Boolean als Übergabeparameter verwenden, handelt es sich immer um eine Wertübergabe (Call by Value) und nicht um eine Referenzübergabe (Call by Reference) wie bei Objektdatentypen. Betrachten Sie dazu folgenden Code:

```
function doppel (n:Number)
{
   n *= 2;
}
var zahl = 2;
doppel (zahl);
trace (zahl);
```

Die Ausgabe des Beispiels lautet 2, da eben nur der *Wert* von zahl an die Funktion übergeben wird.

Nach der Definition der Funktion wird eine Instanz von Auto mit einer Tankkapazität von 50 l und einem Verbrauch von 10 l pro 100 km erzeugt:

```
var auto:Auto = new Auto (50, 10);
```

Der Rest des Codes ruft abwechselnd die Methoden der Instanz und die lokale Funktion `auskunft()` auf. Beispiel:

```
auto.fahren (250);
trace ("Nach 250 km Fahrt:");
auskunft(auto);
```

Wie Sie bereits am Beispiel der einzelnen `Auto`-Instanz sehen können, ist der objektorientierte Ansatz erheblich übersichtlicher und in der Folge weniger fehleranfällig als die imperative Variante. Endgültig klar macht dies die (hier vollständige) Neuimplementierung des Array-Beispiels unter Verwendung der Klasse:

```
function auskunft (a:Auto, n:Number)
{
   trace ("Das Auto " + n + " hat " + a.getTankFuellung() +
          " l im Tank und " + a.getKmStand() + " km auf dem Tacho.");
}

var autos = new Array();
autos[0] = new Auto (35, 6);
autos[1] = new Auto (50, 10);
autos[2] = new Auto (45, 15);
for (var i:Number = 0; i < 3; i++) {
   auskunft (autos[i], i);
   autos[i].fahren (250);
   auskunft (autos[i], i);
   autos[i].tanken (10);
   auskunft (autos[i], i);
}
```

Beachten Sie, dass ein Array stets eine Sammlung von Elementen *beliebiger* Datentypen sein kann – Sie können weder dem Array insgesamt noch seinen einzelnen Elementen einen festen Datentyp zuordnen.

Hier der Vollständigkeit halber die Ausgabe des Beispiels:

```
Das Auto 0 hat 35 l im Tank und 0 km auf dem Tacho.
Das Auto 0 hat 20 l im Tank und 250 km auf dem Tacho.
Das Auto 0 hat 30 l im Tank und 250 km auf dem Tacho.
Das Auto 1 hat 50 l im Tank und 0 km auf dem Tacho.
Das Auto 1 hat 25 l im Tank und 250 km auf dem Tacho.
Das Auto 1 hat 35 l im Tank und 250 km auf dem Tacho.
Das Auto 2 hat 45 l im Tank und 0 km auf dem Tacho.
Das Auto 2 hat 7.5 l im Tank und 250 km auf dem Tacho.
Das Auto 2 hat 17.5 l im Tank und 250 km auf dem Tacho.
```

Klassen entwerfen und programmieren

Nach dem praktischen Einstieg im ersten Abschnitt werden einige Aspekte der objektorientierten ActionScript-Programmierung hier vertieft. Zunächst werden die bereits beschriebenen Aspekte von Klassen vertieft und einige weitere erläutert. Als Nächstes geht es um die Vererbung, das heißt die Ableitung von Klassen, wobei nur

Änderungen und Ergänzungen implementiert werden müssen. Ein interessantes Konzept ist auch die objektorientierte Behandlung von Ausnahmen, das heißt Fehlerzuständen. Schließlich wird noch beschrieben, wie Sie Klassen aus anderen Verzeichnissen einbinden können.

Klassen und ihre Elemente

Die grundlegenden »Zutaten« für eine Klasse haben Sie bereits kennen gelernt: Eigenschaften, Konstruktor und Methoden. Diese Elemente – und diverse interessante Abwandlungen davon – werden hier noch einmal genau beschrieben, größtenteils wieder anhand des `Auto`-Beispiels.

Die Klassendeklaration

Eine ActionScript-Klasse wird stets mit Hilfe der Schreibweise `class` *`Klassenname`* definiert, gefolgt von einem Block mit den Klassenelementen. Bei der Klasse `Auto` sieht dies so aus:

```
class Auto
{
   // Klassenelemente ...
}
```

Wenn eine Klasse durch Vererbung von einer anderen abgeleitet wird, muss der Zusatz `extends` *`Elternklasse`* verwendet werden. Weiter unten im Abschnitt über Vererbung wird die Klasse `LKW` von `Auto` abgeleitet; der Rahmen ihrer Deklaration sieht so aus:

```
class LKW extends Auto
{
   // Elemente, die LKW von Auto unterscheiden
}
```

Wie bereits erwähnt, müssen ActionScript-Klassendefinitionen in einer separaten *.as*-Datei gespeichert werden, die genauso heißen muss wie die Klasse (einschließlich Groß- und Kleinschreibung) und standardmäßig im gleichen Verzeichnis liegt wie die Flash-Filme, die darauf zugreifen.



Um Klassen aus anderen Verzeichnissen einzubinden, können Sie diese unter *Film exportieren* beziehungsweise auf der Registerkarte *Flash* in den *Einstellungen für Veröffentlichungen* wählen: Klicken Sie neben der Auswahl *ActionScript 2.0* auf *Einstellungen*, und wählen Sie Ihren *Classpath*. Mit Hilfe der »Plus«-Schaltfläche können Sie Pfade manuell eingeben, während das Fadenkreuz einen Ordnerwahldialog öffnet.

Ein spezieller Modifikator für Klassendefinitionen ist `dynamic`: Er erlaubt es, Klassen zur Laufzeit um beliebige Elemente zu erweitern. Im Wesentlichen existiert `dynamic`,

um Instanzen eingebauter Klassen wie `MovieClip`, `Array` oder `Object` nach Bedarf um Elemente zu ergänzen – eine Anweisung wie `meinClip.variable=Wert` macht sich genau dies zunutze.

Sie können aber auch selbst dynamische Klassen erstellen und verwenden. Um ein schnelles Beispiel zu erhalten, wird hier kurz die Vererbung vorweggenommen, um eine dynamische, aber ansonsten völlig identische Ableitung von `Auto` zu erstellen (nur der Konstruktor muss neu geschrieben werden, da er nicht automatisch übernommen wird). Beginnen Sie eine neue ActionScript-Datei, und schreiben Sie folgenden Code hinein:

```
dynamic class DynAuto extends Auto
{
   // Konstruktor
   public function DynAuto (m:Number, v:Number)
   {
      // Konstruktor der Elternklasse aufrufen
      super (m, v);
   }
   // Ansonsten keine Änderungen/Ergänzungen
}
```

Speichern Sie die Klasse unter dem Namen *DynAuto.as* im gleichen Verzeichnis wie ihre Elternklasse *Auto.as*. Anschließend können Sie einen neuen Flash-Film mit folgendem Bild-Skript erstellen, um die neue Dynamik auszuprobieren:

```
// Eine DynAuto-Instanz erstellen
var a:DynAuto = new DynAuto (50, 10);
// Dynamisch eine neue Eigenschaft hinzufügen
a.farbe = "gruen";
// Eigenschaft durch Ausgabe testen
trace ("Das Auto ist " + a.farbe);
// Eine neue Methode als anonyme Funktion hinzufügen
a.setVerbrauch = function (l:Number):Void {
   this.verbrauch = l;
}
// Die neue Methode aufrufen
a.setVerbrauch (11);
// Erfolg des Methodenaufrufs durch Ausgabe testen
trace ("Das Auto verbraucht jetzt " + a.getVerbrauch() + " l");
```

Die Ausgabe des Beispiels sieht erwartungsgemäß so aus:

```
Das Auto ist gruen
Das Auto verbraucht jetzt 11 l
```



Der Einsatz dynamischer Klassen widerspricht dem Paradigma der Kapselung – wie bei der imperativen Programmierung können Sie die Datenstruktur von einer beliebigen Stelle in Ihren Skripten manipulieren. Es handelt sich um eine Art »schnellen Hack«, der in ordentlichem objektorientiertem Design nichts verloren hat. Wenn Sie Ihre Klassen elegant um zusätzliche Elemente erweitern möchten, verwenden Sie die weiter unten besprochene Vererbung.

Eigenschaften

Mit Hilfe von *Eigenschaften* – den innerhalb einer Klasse deklarierten Variablen – wird die Datenstruktur der Klasse und ihrer künftigen Instanzen entworfen. In den meisten Fällen sollten Eigenschaften mit Hilfe des Schlüsselwortes `private` unzugänglich gemacht werden; die erwünschten Zugriffe erfolgen dann im jeweiligen Einzelfall durch passende Accessor-Methoden.

Die Klasse `Auto` definiert insgesamt vier Eigenschaften:

```
private var kmStand:Number;
private var tankFuellung:Number;
private var maxTankFuellung:Number;
private var verbrauch:Number;
```

Der konkrete Verwendungszweck dieser Eigenschaften wurde bereits besprochen. Ihre Anfangswerte erhalten sie durch Aufruf des Konstruktors; dies ist die übliche Methode. Es wäre in gewisser Weise unlogisch, den Eigenschaften schon bei der Deklaration Anfangswerte zuzuweisen, selbst dann, wenn sie durch die Parameter des Konstruktoraufrufs nicht beeinflusst würden (dies trifft im vorliegenden Beispiel nur für `kmStand` zu). Denn die Eigenschaften gehören *nicht* zur Klasse, sondern zur jeweiligen Instanz. Ihre Existenz beginnt daher erst durch die Instantiierung, die durch den Operator `new()` den Konstruktor aufruft. Daher werden solche »normalen« Eigenschaften genauer als *Instanzeigenschaften* bezeichnet.

Wenn Sie unbedingt möchten, können Sie Eigenschaften auch `public` deklarieren; sie stehen dann direkt über die Instanz zum Auslesen und Ändern zur Verfügung. Wenn überhaupt, sollte dies aber nur mit eher unwichtigen Eigenschaften geschehen, die keine »gefährlichen« Werte annehmen können. Beim Auto ließe sich beispielsweise eine relativ harmlose String-Eigenschaft für dessen Farbe hinzufügen:

```
// Neue Eigenschaft
public var farbe:String;
```

Diese Eigenschaft könnte direkt über die Instanz gesetzt werden. Beispiel:

```
var auto:Auto = new Auto (40, 8);
auto.farbe = "rot";
```

Eine ganz besondere Bedeutung haben Eigenschaften, die mit dem Schlüsselwort `static` deklariert werden: Es handelt sich um *Klasseneigenschaften* im Gegensatz zu den oben beschriebenen Instanzeigenschaften; sie werden auch als statische Eigenschaften bezeichnet. Die Besonderheit besteht darin, dass Sie über die Schreibweise `Klasse.`*`Eigenschaft`* darauf zugreifen können, ohne eine Instanz zu erzeugen. Damit dies nützlich ist, sollten Klasseneigenschaften in aller Regel `public` sein. Genutzt werden sie typischerweise im Sinne von Konstanten – im vorigen Kapitel wurde beispielsweise die Klasseneigenschaft `Math.PI` erwähnt, die den Wert von π enthält.

Da es für ein Auto keine allgemein nützliche Konstante gibt, sehen Sie hier als Beispiel eine ganz neue Klasse, die nur aus statischen Eigenschaften besteht:

```
class Farbe {
   public static var SCHWARZ:Number = 0x000000;
   public static var ROT:Number = 0xFF0000;
   public static var GRUEN:Number = 0x00FF00;
   public static var BLAU:Number = 0x0000FF;
   public static var GELB:Number = ROT | GRUEN;
   public static var MAGENTA:Number = ROT | BLAU;
   public static var CYAN:Number = GRUEN | BLAU;
   public static var WEISS:Number = ROT | GRUEN | BLAU;
}
```

Die Elemente dieser Klasse sind numerische RGB-Farbwerte. Sie können sie in der Form `Farbe.ROT` oder `Farbe.SCHWARZ` verwenden; beispielsweise für die im vorigen Kapitel beschriebene Einfärbung von Movieclips.



Auf statische Eigenschaften darf innerhalb der Klassendefinition *nicht* mittels `this` zugegriffen werden. Wie bereits erwähnt, bezieht sich `this` auf die aktuelle Instanz, aber Klasseneigenschaften sind keiner Instanz zugeordnet. Zur Unterscheidung können Sie aber auch innerhalb der Klasse *`Klassenname.Eigenschaft`* schreiben.

Konstruktoren

Der *Konstruktor* ist eine spezielle Funktion, die automatisch beim Erstellen einer neuen Instanz mittels `new()` aufgerufen wird. Sie darf keine Datentypangabe besitzen, da ihr Datentyp eben die Klasse selbst ist. Die Aufgabe eines Konstruktors ist die Initialisierung der Instanz; im Wesentlichen geht es darum, den Eigenschaften ihre Anfangswerte zuzuweisen. Häufig – wie beim Auto-Beispiel – werden einige von ihnen vom Konstruktor als Parameter erwartet:

```
public function Auto (m:Number, v:Number)
{
   this.maxTankFuellung = m;
   this.verbrauch = v;
   // Zu Anfang: km-Stand 0, Tank voll
   this.kmStand = 0;
   this.tankFuellung = this.maxTankFuellung;
}
```

Wenn Sie gar keinen Konstruktor definieren, stellt ActionScript einen (leeren) Standard-Konstruktor bereit. Falls Sie dies verhindern möchten (und keinen anderen Konstruktor definiert haben), können Sie den Konstruktor einfach explizit `private` setzen. Einen Sinn ergäbe dies beispielsweise bei der oben beschriebenen Klasse `Farbe`, die rein statisch verwendet wird:

```
// Instanzerzeugung verbieten - Konstruktor private setzen
private function Farbe()
{
}
```

Der folgende Versuch scheitert dann mit der angegebenen Fehlermeldung:

```
var f:Farbe = new Farbe();

**Fehler** Szene=Szene 1, Ebene=Ebene 1, Bild=1:Zeile 9:
 Das Mitglied ist privat, und der Zugriff darauf ist nicht möglich.
     var f:Farbe = new Farbe();
```

Methoden

Die innerhalb einer Klasse definierten Funktionen werden *Methoden* genannt. Genau wie bei den Eigenschaften lassen sich *Instanzmethoden* und (durch `static` gekennzeichnete) *Klassenmethoden* unterscheiden. Der Zugriff auf Instanzmethoden erfolgt mittels *`Instanz.Methode`*`()`; ein Beispiel wäre der folgende Code, der Methoden einer `Auto`-Instanz aufruft:

```
// Instanz erzeugen
var a:Auto = new Auto (35, 7);
// Methoden aufrufen
a.fahren (300);
a.tanken (10);
```

Für Methoden sollten Sie stets einen Datentyp angeben. Eine Methode, die keinen Wert zurückgibt, erhält dabei den speziellen Datentyp `Void`, wie etwa die – bereits ausführlich besprochenen – `Auto`-Methoden `fahren()` und `tanken()`.

In Einzelfällen kann es sinnvoll sein, Methoden nicht als `public`, sondern als `private` zu definieren, nämlich immer dann, wenn sie Hilfsaufgaben für andere Methoden der Klasse wahrnehmen, aber nicht von Instanzen aus aufgerufen werden sollen. Das folgende Beispiel verlagert den Dreisatzterm zur Berechnung des Verbrauchs pro Strecke in eine private Methode:

```
private function verbrauchProStrecke (km:Number):Number
{
   return km * this.verbrauch / 100;
}
```

Natürlich müsste die Methode `fahren()` umgeschrieben werden, um davon Gebrauch zu machen:

```
public function fahren (km:Number):Void {
   // Nur fahren, falls das Benzin reicht
   if (this.verbrauchProStrecke (km) <= this.tankFuellung) {
      kmStand += km;
      tankFuellung -= this.verbrauchProStrecke (km);
   }
}
```

Eine noch allgemeinere Aufgabe besitzen statische Methoden oder Klassenmethoden: Sie können auch von außen als Element der Klasse aufgerufen werden, ohne eine Instanz zu erzeugen. Die im vorigen Kapitel angesprochenen Methoden der Klasse `Math` wie `Math.floor()` sind ein bekanntes Beispiel.

Für das Auto-Beispiel ließe sich auf diese Weise etwa eine ganz allgemeine Rechenmethode schreiben, die den Kraftstoffverbrauch pro 100 km aus einer angegebenen Strecke und einer tatsächlich verbrauchten Menge Kraftstoff ermittelt. Diese Klassenmethode könnte beispielsweise so aussehen:

```
public static function berechneVerbrauch (km:Number, l:Number):Number
{
   return l * 100 / km;
}
```

Diese Methode könnten Sie einsetzen, um einen hypothetischen Kraftstoffverbrauch zu berechnen, ohne eine `Auto`-Instanz zu erzeugen. Beispiel:

```
// 320 km gefahren, 30 l verbraucht
trace (Auto.berechneVerbrauch (320, 30));
```

Das Ergebnis ist ein Verbrauch von 9.375 l auf 100 km.

Accessor-Methoden

Die bereits erwähnten *Accessor-Methoden* dienen dem direkten Zugriff auf ausgewählte Instanzeigenschaften. Man unterscheidet *Getter-Methoden*, die einfach den Wert einer Eigenschaft zurückgeben, und *Setter-Methoden*, die der Eigenschaft einen neuen, angegebenen Wert zuweisen. Accessor-Methoden bieten die Möglichkeit, Eigenschaften auszulesen oder zu modifizieren, obwohl sie (empfehlenswerterweise) als `private` deklariert wurden. Da Sie sich selbst aussuchen können, für welche Eigenschaften Sie eine Getter- und/oder Setter-Methode bereitstellen, widerspricht dieser Weg nicht dem Konzept der Kapselung.

Getter-Methoden haben Sie bereits kennen gelernt; die Klasse `Auto` enthält eine von ihnen für jede der vier Instanzeigenschaften. Der Datentyp einer Getter-Methode muss demjenigen der jeweiligen Eigenschaft entsprechen; die einzige Anweisung lautet in der Regel `return this.`*`Eigenschaft`*. Hier als Beispiel noch einmal die Methode `getKmStand()`:

```
public function getKmStand():Number
{
   return this.kmStand;
}
```

Eine Setter-Methode ist in der Klasse `Auto` bisher nicht enthalten, da maximale Tankfüllung und Verbrauch konstant bleiben sollen, während Kilometerstand und aktuelle Tankfüllung nur durch den Aufruf der Methoden `fahren()` beziehungsweise `tanken()` modifiziert werden sollen. Eine einigermaßen plausible Setter-Methode wäre allerdings `setVerbrauch()`, weil Autos dazu neigen, allmählich mehr Kraftstoff zu verbrauchen, wenn sie älter werden. Diese Methode könnte so aussehen:

```
public function setVerbrauch (v:Number):Void
{
   this.verbrauch = v;
}
```

Wie hier gezeigt, benötigen Setter-Methoden einen Parameter mit dem Datentyp der zu ändernden Eigenschaft; ihr eigener Typ ist dagegen Void, weil sie keinen Wert zurückgeben müssen. Auch Setter enthalten normalerweise nur eine Anweisung, nämlich eine Wertzuweisung, die die Eigenschaft auf den übergebenen Wert setzt. Zusätzlich könnte man aber auch eine Plausibilitätskontrolle einführen, die nur bestimmte Werte zulässt. Beispielsweise ließe sich `setVerbrauch()` folgendermaßen ergänzen, damit sich der Verbrauch erhöhen, aber nicht vermindern lässt:

```
public function setVerbrauch (v:Number):Void
{
   // Nur ändern, falls v > aktueller Verbrauch
   if (v > this.verbrauch) {
      this.verbrauch = v;
   }
}
```



In der Praxis sollte ein unzulässiger Wert nicht einfach kommentarlos verworfen werden, sondern zu einem definierten Fehler führen. Näheres dazu finden Sie weiter unten im Abschnitt *Ausnahmenbehandlung*.

Vererbung

Eine der interessantesten Fähigkeiten objektorientierter Programmiersprachen ist die Spezialisierung von Klassen durch die so genannte *Vererbung*. Sie ermöglicht es Ihnen, »das Rad nicht neu erfinden« zu müssen – einmal gefundene Lösungen lassen sich in jeweils abgewandelter oder ergänzter Form wiederverwenden.

Formal funktioniert Vererbung sehr einfach: Bei der Definition der Klasse wird das Schlüsselwort `extends` und der Name der Elternklasse hinzugefügt. Hier als Beispiel der Rahmen der von `Auto` abgeleiteten Klasse `LKW`, die gleich näher beschrieben wird:

```
class LKW extends Auto
{
   // Änderungen und Ergänzungen, die aus dem Auto einen LKW machen
}
```

Instanzen der abgeleiteten Klasse verfügen über sämtliche öffentlichen Eigenschaften und Methoden der Elternklasse. Wichtig ist allerdings, dass Sie den Konstruktor neu schreiben; er wird nicht automatisch vererbt. Glücklicherweise können Sie innerhalb des neuen Konstruktors aber denjenigen der Elternklasse aufrufen, indem Sie `super()` schreiben.

Sogar private Eigenschaften und Methoden der Elternklasse stehen innerhalb der abgeleiteten Klasse zur Verfügung. Dies ist eine Besonderheit von ActionScript; andere objektorientierte Sprachen wie etwa Java vererben private Elemente nicht. Dafür bieten diese neben `public` und `private` eine dritte Geheimhaltungsstufe namens `protected`, die genau dafür zuständig ist.

Bevor es mit der Theorie weitergeht, hier das vollständige Listing der Klasse `LKW`, die Sie in einer Datei namens *LKW.as* im selben Verzeichnis wie *Auto.as* speichern müssen:

```
class LKW extends Auto
{
   // Neue Eigenschaften
   private var maxLadung:Number;
   private var ladung:Number;

   // Konstruktor
   public function LKW (m:Number, v:Number, l:Number)
   {
      // Zunächst ein Auto mit der gewünschten max.
      // Tankfüllung und Verbrauch erstellen
      super (m, v);
      // Ladekapazität und aktuelle Ladung einstellen
      this.maxLadung = l;
      this.ladung = 0;
   }

   // Neue Methoden
   public function beladen (l:Number)
   {
      // Nur hinzuladen, falls Kapazität genügt
      if (this.ladung + l <= maxLadung) {
         this.ladung += l;
      }
   }

   public function entladen (l:Number)
   {
      // Nur entladen, falls so viel vorhanden
      if (this.ladung - l >= 0) {
         this.ladung -= l;
      }
   }

   // Neue Getter-Methoden
   public function getLadung():Number
   {
      return this.ladung;
   }

   public function getMaxLadung():Number
   {
      return this.maxLadung;
   }
}
```

Wie Sie sehen, verfügt ein LKW über zwei neue Eigenschaften, die seine Ladung beziehungsweise Ladekapazität (in Tonnen) repräsentieren. Die Methoden `bela-`

den() und entladen() fügen Ladung hinzu beziehungsweise entfernen sie – wie gehabt jeweils mit Plausibilitätsprüfung. Jede der beiden Eigenschaften erhält zudem ihre eigene Getter-Methode.

Interessant ist der Konstruktor. Er nimmt drei Parameter entgegen, nämlich Tankkapazität, Verbrauch und maximales Ladungsgewicht. Die ersten beiden Werte entsprechen den Eigenschaften der Elternklasse Auto, deshalb werden sie an deren Konstruktor weitergereicht:

```
super (m, v);
```

Der Rest – die Initialisierung der neuen Eigenschaften – muss dagegen an Ort und Stelle erledigt werden, weil die Klasse Auto diese Eigenschaften nicht besitzt:

```
this.maxLadung = l;
this.ladung = 0;
```

Der Rest der Klassendefinition ist kaum der Rede wert: Die beiden Methoden beladen() und entladen() fügen die angegebene Menge an Ladung hinzu beziehungsweise entfernen sie; zwei Getter für die neuen Eigenschaften schließen den Code ab.

Eine LKW-Instanz wird nach dem üblichen Schema erstellt. Beispiel:

```
var lkw:LKW = new LKW (100, 18, 7);
```

Auch der Methodenaufruf erfolgt wie gewohnt. Natürlich können Sie ohne jeden Unterschied sowohl die Methoden der Klasse als auch der Elternklasse aufrufen, wie das folgende Beispiel zeigt:

```
lkw.beladen (4);
lkw.fahren (100);
trace ("Der LKW hat " + lkw.getKmStand() + " km auf dem Tacho und " +
       lkw.getTankFuellung() + " l im Tank.\nDie Ladung beträgt " +
       lkw.getLadung() + " t.");
```

Die Ausgabe lautet erwartungsgemäß:

```
Der LKW hat 100 km auf dem Tacho und 82 l im Tank.
Die Ladung beträgt 4 t.
```

Etwas überraschender ist vielleicht, dass Sie eine LKW-Instanz verwenden können, wo eine Auto-Instanz erwartet wird – beispielsweise in folgender bereits beschriebenen Version der Funktion auskunft():

```
function auskunft (a:Auto):Void
{
   trace ("Tankfüllung: " + a.getTankFuellung() + ", " +
          "Kilometerstand: " + a.getKmStand());
}

// LKW-Instanz erstellen
var lkw:LKW = new LKW (120, 22, 14);
// Auskunft erhalten
auskunft (lkw);
```

Dies offenbart ein allgemeines Merkmal der Vererbung: Jede Instanz einer Klasse ist implizit auch Instanz ihrer Elternklasse sowie aller übergeordneten Klassen. Diese Tatsache wird in der Objektorientierung als »IS-A«-Beziehung bezeichnet: Jeder LKW »ist ein« Auto.

Eine völlig andere Relation ist übrigens die »HAS-A«-Beziehung – sie liegt vor, wenn eine Klasse eine Instanzeigenschaft enthält, die Instanz einer anderen Klasse ist. Beispielsweise könnte man einen Tank als separate Klasse modellieren und in verschiedene Fahrzeuge »einbauen«.

Vererbung funktioniert übrigens sowohl beliebig tief (Sie können von jeder abgeleiteten Klasse wiederum Klassen ableiten) als auch beliebig breit (von jeder Klasse können beliebig viele Klassen direkt abgeleitet werden). Als Beispiel sehen Sie im Folgenden noch eine Erweiterung der Klasse `LKW` – `AnhaengerLKW` – sowie eine weitere Ableitung von `Auto` namens `PKW`. Die resultierende Klassenhierarchie lässt sich durch ein UML-Klassendiagramm wie in Abbildung 7-1 darstellen.

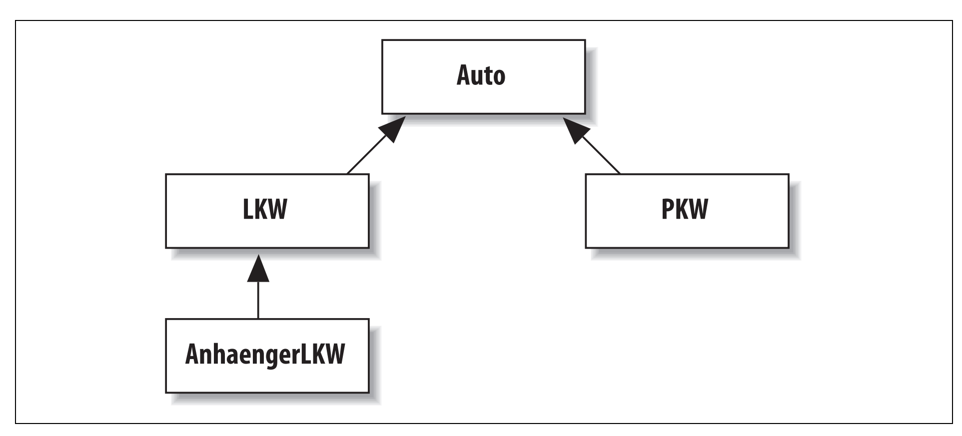


Abbildung 7-1: UML-Klassendiagramm der Klasse Auto und ihrer Ableitungen

Hier zunächst die Klasse `AnhaengerLKW`, die eine `Boolean`-Eigenschaft namens `anhaenger` und entsprechende Methoden hinzufügt:

```
class AnhaengerLKW extends LKW
{
   // Neue Eigenschaft: Anhänger
   private var anhaenger:Boolean;

   // Konstruktor
   public function AnhaengerLKW (m:Number, v:Number, l:Number)
   {
      // LKW-Konstruktor aufrufen
      super (m, v, l);
      // Anhänger ist standardmäßig nicht angehängt
      this.anhaenger = false;
   }
```

```
    // Setter zum An-/Abhängen des Anhängers
    public function setAnhaenger (a:Boolean):Void
    {
       this.anhaenger = a;
    }

    // Getter zum Ermitteln des Anhänger-Status'
    public funciton getAnhaenger():Boolean
    {
       return this.anhaenger;
    }
 }
```

Die von `Auto` abgeleitete Klasse `PKW` definiert eine feste Anzahl von Sitzplätzen sowie eine variable Insassenzahl:

```
class PKW extends Auto
{
   // Neue Eigenschaften
   private var sitze:Number;
   private var insassen:Number;

   // Konstruktor
   public function PKW (m:Number, v:Number, s:Number)
   {
      // Zunächst ein Auto mit der gewünschten max.
      // Tankfüllung und Verbrauch erstellen
      super (m, v);
      // Anzahl der Sitze und aktuelle Insassenzahl einstellen
      this.sitze = s;
      this.insassen = 0;
   }

   // Neue Methoden
   public function einsteigen (p:Number)
   {
      // Nur einsteigen, falls Sitze frei
      if (this.insassen + p <= sitze) {
         this.insassen += p;
      }
   }

   public function aussteigen (p:Number)
   {
      // Nur abziehen, falls so viele vorhanden
      if (this.insassen - p >= 0) {
         this.insassen -= p;
      }
```

```
        }

        // Neue Getter-Methoden
        public function getInsassen():Number
        {
           return this.insassen;
        }

        public function getSitze():Number
        {
           return this.sitze;
        }
    }
```

Nach den bisherigen Anleitungen in diesem Kapitel können Sie diese neuen Klassen leicht selbst ausprobieren.

Ausnahmenbehandlung

Wenn innerhalb von Klassen Fehler auftreten – zum Beispiel aufgrund falscher Konstruktorparameter –, ist es wünschenswert, diese an der Stelle zu bearbeiten, an der die jeweilige Instanz erzeugt beziehungsweise verwendet wird. Zu diesem Zweck verwenden objektorientierte Programmiersprachen so genannte *Ausnahmen*. Diese werden innerhalb von Klassen mit dem Schlüsselwort `throw` ausgelöst. Instanzen einer Klasse, die Ausnahmen auslöst, können von einem `try{ }`-Block umschlossen werden; anschließend wird `catch()` verwendet, um die entsprechende Ausnahme abzufangen.

`throw` muss eine (meist anonyme) Instanz der Standardklasse `Error` auswerfen; wenn Sie möchten, können Sie auch eine eigene, davon abgeleitete Klasse verwenden. Das Einzige, was eine solche Fehlerklasse benötigt, ist eine öffentliche `String`-Eigenschaft namens `message`. Das folgende Beispiel leitet eine Klasse namens `KraftstoffMangelException` von `Error` ab; sie soll in der unten vorgestellten Neufassung der Klasse `Auto` ausgelöst werden, wenn der Kraftstoff nicht ausreicht, um die gewünschte Anzahl von Kilometern zu `fahren()`:

```
class KraftstoffMangelException extends Error
{
   public var message:String = "Der Kraftstoff reicht nicht.";
}
```

Speichern Sie diese Klasse als *KraftstoffMangelException.as* in demselben Verzeichnis wie `Auto`. Für den anderen möglichen Fehler – den Versuch, zu viel zu tanken – wird übrigens ein einfacher `Error` mit angepasstem Meldungstext verwendet, um auch dieses Verfahren zu demonstrieren. Hier die komplette Neuimplementierung von `Auto` mit Ausnahmen:

```
class Auto {
   // private Eigenschaften
   private var kmStand:Number;
```

```
private var tankFuellung:Number;
private var maxTankFuellung:Number;
private var verbrauch:Number;

// Konstruktor
public function Auto (m:Number, v:Number) {
   this.maxTankFuellung = m;
   this.verbrauch = v;
   // Zu Anfang: km-Stand 0, Tank voll
   this.kmStand = 0;
   this.tankFuellung = this.maxTankFuellung;
}

// Private Rechenmethode
private function verbrauchProStrecke (km:Number):Number
{
     return km * this.verbrauch / 100;
}

// Öffentliche Methoden
public function fahren (km:Number):Void {
   // Fehler erzeugen, falls das Benzin nicht reicht
   if (this.tankFuellung < this.verbrauchProStrecke (km)) {
      throw new KraftstoffMangelException();
   } else {
      kmStand += km;
      tankFuellung -= this.verbrauchProStrecke (km);
   }
}

public function tanken (l:Number):Void {
   // Fehler erzeugen, falls der Tank zu voll würde
   if (this.tankFuellung + l > this.maxTankFuellung) {
      throw new Error ("So viel passt nicht in den Tank.");
   } else {
      tankFuellung += l;
   }
}

// Accessor-Methoden (nur Getter)
public function getKmStand():Number {
   return this.kmStand;
}

public function getTankFuellung():Number {
   return this.tankFuellung;
}

public function getMaxTankFuellung():Number {
   return this.maxTankFuellung;
}

public function getVerbrauch():Number {
```

```
        return this.verbrauch;
    }
}
```

Hier ein einfaches Verwendungsbeispiel, das beide Methoden aufruft und die möglicherweise auftretenden Ausnahmen abfängt:

```
var auto:Auto = new Auto (50, 10);
// 200 km fahren, falls das Benzin reicht
try {
   auto.fahren (200);
} catch (e:KraftstoffMangelException) {
   trace (e.message);
}
// 30 l tanken, falls sie in den Tank passen
try {
   auto.tanken (30);
} catch (e:Error) {
   trace (e.message);
}
```

Wichtiger als bei diesem Beispiel mit konstanten Werten ist das Abfangen von Ausnahmen natürlich, wenn Sie unvorhersagbare Werte verwenden – beispielsweise Benutzereingaben oder Zufallszahlen.

Ausblick: ActionScript 3.0



Die Flash-Version 9.0 soll mit der nächsten Generation von ActionScript ausgestattet werden. Ein erster Entwurf von ActionScript 3.0 existiert bereits. Ein Compiler dafür ist in der Entwickler-Vorschau von Macromedia/Adobe FlexBuilder 2.0 eingebaut; diese professionelle (und in der künftigen Endversion sehr teure) Serversoftware enthält außerdem den Flash Player 8.5, der ActionScript 3-Code ausführen kann. Die Beta-Version von Flex 2 können Sie unter *http://www.macromedia.com/go/labs_flex2_downloads* herunterladen.

Es würde zu weit führen, hier praktisch auf die Neuerungen von ActionScript 3 einzugehen. Wenn Sie Interesse haben, halten Sie Ausschau nach dem Buch *Essential ActionScript 3.0* von Colin Moock, das demnächst bei O'Reilly erscheint. Bis dahin können Sie schon einmal einen Blick in das in Arbeit befindliche Flex-Handbuch *Programming ActionScript 3.0* (*http://www.macromedia.com/go/programmingAS3*) sowie in die ActionScript 3-Referenz (*http://www.macromedia.com/go/AS3LR*) werfen. Hier finden Sie nur einen kurzen Überblick über einige wichtige Funktionen der neuen Version, ergänzt durch einige Beispiele.

→

Eine der wichtigsten Neuerungen ist der alternative numerische Datentyp `int`, der im Gegensatz zu `Number` nur ganze Zahlen (Integer) speichert. Durch die Kapazität von 32 Bit beträgt der Wertebereich -2.147.483.648 bis +2.147.483.647; die Verarbeitung erfolgt wesentlich schneller als bei Fließkommazahlen. Der alternative Typ `uint` (Unsigned Integer) erlaubt nur positive Ganzzahlen zwischen 0 und 4.294.967.295.

Eine lang erwartete Erweiterung ist die Unterstützung regulärer Ausdrücke gemäß ECMAScript 3-Spezifikation. Reguläre Ausdrücke (Regular Expressions oder kurz Regexp) bieten eine vor allem in der UNIX-Welt bekannte, sehr leistungsfähige Syntax für Suchmuster. Das folgende sehr kurze Beispiel überprüft, ob die Variable `nummer` eine gültige ISBN-Nummer (mit Bindestrichen oder auch ohne) enthält:

```
// Die ISBN des vorliegenden Buchs:
var nummer = "Praxiswissen Flash 8: 3-89721-450-4";
if (nummer.match ("[0-9]\-?[0-9]{5}\-?[0-9]{3}\-?[0-9X]") {
   trace ("Gültige ISBN gefunden.");
} else {
   trace ("Keine ISBN enthalten.");
}
```

Die neue `String`-Methode `match()` überprüft, ob der angegebene reguläre Ausdruck auf den String zutrifft. Der Ausdruck selbst bedarf einiger Erklärung: Eckige Klammern umschließen eine Gruppe von Zeichen, von denen eines an der angegebenen Stelle stehen muss. Der Bindestrich dient dabei der Bereichsbildung – `[09]` steht also für eines der beiden Zeichen 0 oder 9, während `[0-9]` eine beliebige Ziffer repräsentiert. Eine Zahl in geschweiften Klammern gibt an, wie oft ein Zeichen oder sonstiges Element vorkommen soll – `[0-9]{5}` bedeutet also beispielsweise fünf Ziffern. Der Bindestrich als gesuchtes Zeichen muss durch einen vorangestellten Backslash (\) gekennzeichnet werden, genau wie alle anderen Zeichen, die in regulären Ausdrücken eine besondere Bedeutung haben (zum Beispiel `. , [ ] { }`). Das Fragezeichen macht das Vorkommen des vorherigen Zeichens oder Blocks optional – der Bindestrich soll also entweder vorkommen oder auch nicht. Der letzte Block, `[0-9X]`, repräsentiert die Prüfsumme: eine beliebige Ziffer oder ein X.

Eine ausführliche Einführung in reguläre Ausdrücke erhalten Sie unter *http://www.galileocomputing.de/openbook/kit/itkomp06001.htm*. Ihre ActionScript 3.0-kompatible Verwendung in JavaScript wird kurz in *http://www.galileocomputing.de/openbook/kit/itkomp19002.htm* angesprochen. Beide Adressen verweisen auf die kostenlos zugängliche Online-Fassung meines Buches *Kompendium der Informationstechnik* (Bonn 2003, Galileo Press).

Als Alternative zu DOM mit erheblich einfacherer Syntax bietet ActionScript 3 Unterstützung für E4X (ECMAScript for XML). Kurz gesagt betrachtet diese API XML-Dokumente als normale Datenstrukturen, die den Zugriff auf verschachtelte XML-Elemente im Format `Element.Unterlement` erlauben.

→

Einige Erweiterungen betreffen die Objektorientierung: Das Schlüsselwort `final` vor einer Klassendeklaration verhindert, dass Klassen von der entsprechenden Klasse abgeleitet werden; bei einer Methode sorgt es entsprechend dafür, dass diese in einer untergeordneten Klasse nicht überschrieben werden kann. Genau wie Methoden können nun auch Klassen `public` oder `private` sein; Letzteres ist allerdings nur in den ebenfalls neu eingeführten verschachtelten Klassen (also Klassen innerhalb einer anderen Klassendefinition) sinnvoll. Zudem wurde eine weitere Geheimhaltungsstufe hinzugefügt: `protected`-Elemente stehen nicht öffentlich zur Verfügung, aber in Instanzen abgeleiteter Klassen. Das Schlüsselwort `package` ermöglicht es schließlich, Klassenbibliotheken zu gemeinsamen Namensräumen zusammenzufassen. Das Ergebnis sind Klassennamen, denen ein durch einen Punkt getrennter Package-Name vorangestellt wird, was unter anderem für die Vermeidung von Namenskonflikten sorgt.

Schließlich gibt es einige Erweiterungen in der Flash Player-API selbst: Die neue Klasse `Sprite` ist weniger aufwändig als `MovieClip` und dient somit als praktischere Basisklasse für UI-Komponenten. Per ActionScript erstellte Objekte benötigen nun keine Tiefenangabe mehr; sie werden automatisch »gestapelt« (mehr zum Problem der Tiefe wird im XML-Beispiel in Kapitel 10 besprochen).

Völlig neue Möglichkeiten ergeben sich durch die Klasse `flash.net.Socket` – Sie können damit Netzwerkverbindungen über viele verschiedene Protokolle herstellen; SWF-Filme können dadurch zum Beispiel mit Mail- oder Webservern kommunizieren.

Ein sehr nettes Spielzeug ist schließlich die Methode `Sound.computeSpectrum()`: Sie liefert ein Array mit dem Frequenzspektrum aller derzeit abgespielten Sounds zurück. Dies ermöglicht die Programmierung beliebiger Sound-Visualisierungselemente.

Praxisbeispiel: Eine Komfort-Datumsklasse

Wie die meisten Programmiersprachen enthält auch ActionScript die Möglichkeit, auf die Systemuhr des Computers zuzugreifen und so Datum und Uhrzeit auszulesen. Dazu dient die ECMAScript-Standardklasse `Date`, die zahlreiche Methoden zum Auslesen der einzelnen Datums- und Uhrzeitkomponenten (Tag, Monat, Stunde, Minute und so weiter) enthält. Um sie zu benutzen, müssen Sie zunächst ein Objekt dieser Klasse erstellen:

```
var d = new Date();
```

Dieses Objekt – hier heißt es `d` – speichert den aktuellen Zeitpunkt gemäß Systemuhr; es läuft nicht selbst wie eine Uhr weiter. Es bietet Ihnen unter anderem folgende Methoden:

- `d.getFullYear()` – vierstellige Jahreszahl
- `d.getMonth()` – Monat (von 0 für Januar bis 11 für Dezember)
- `d.getDate()` – Tag im Monat (diesmal korrekt von 1 bis 31)

- `d.getDay()` – numerischer Wochentag (0=Sonntag, 1=Montag, ..., 6=Samstag)
- `d.getHours()` – Stunde (0 bis 23)
- `d.getMinutes()` – Minute (0 bis 59)
- `d.getSeconds()` – Sekunde (0 bis 59)
- `d.toString()` – das komplette, formatierte Datum im so genannten RFC-1123-Format: `Tue Nov 29 08:49:02 GMT+0200 2005`

Fast alle `get*()`-Methoden besitzen übrigens auch eine zugehörige `set*()`-Methode, mit der Sie »die Uhr verstellen« können. Selbstverständlich gibt es keine Methode namens `setDay()`, da der Wochentag vom Datum abhängt.

Das Praxisbeispiel dieses Kapitels ist die von `Date` abgeleitete Klasse `CustomDate`. Sie erweitert das Angebot der Standardklasse um zahlreiche interessante Methoden: Es werden (mehrsprachige) Textdarstellungen von Wochentagen und Monaten hinzugefügt. Außerdem unterstützt die Klasse so genannte `strftime()`-Formate; diese Funktion aus der Standardbibliothek der Programmiersprache C setzt eine Reihe von `%`-Platzhaltern in die Bestandteile von Datum und Uhrzeit um.

Die Beschreibung und Benutzerdokumentation zu `CustomDate` finden Sie in Form ausführlicher Kommentare innerhalb des nachfolgenden Listings. Der Kommentarstil entspricht dem für Java-Klassen entworfenen Dokumentations-Tool `javadoc`. Auf der Website zum Buch finden Sie Links zu einigen Tools, die aus solchen Kommentaren automatisch eine HTML-Dokumentation generieren.

```
class CustomDate extends Date {
   /**
    * CustomDate, Version 0.1
    * Erweitert die eingebaute Klasse Date
    * um Textdarstellungen von Wochentag und Monat
    * sowie um strftime()-Formate.
    *
    * Siehe: http://buecher.lingoworld.de/flash8/classes
    *
    * @author: Sascha Kersken
    * @version: 0.1.0
    */

   private function pre0(n:Number):String {
      // 0 voranstellen, falls n < 10
      return n < 10 ? "0" + n : n;
   }

   /**
    * getInternationalDayName(), Methode
    *
    * Lokalisierter Wochentagname nach ISO-Sprachkürzel
    * (zz. sind nur Deutsch und Englisch implementiert;
    *  Englisch ist Standard)
    *
```

```
 * @param lang  ISO-Sprachkürzel wie en, de usw.
 */
public function getInternationalDayName(lang:String):String {
   var wdays:Array = new Array();
   switch (lang) {
   case "de" :
      wdays.push("Sonntag", "Montag", "Dienstag", "Mittwoch",
                 "Donnerstag", "Freitag", "Samstag");
      break;
   case "en" :
   default :
      wdays.push("Sunday", "Monday", "Tuesday", "Wednesday",
                 "Thursday", "Friday", "Saturday");
   }
   return wdays[this.getDay()];
}

/**
 * getDayName(), Methode
 *
 * Gibt den englischen Namen des Wochentags zurück
 *
 */
public function getDayName():String {
   return this.getInternationalDayName("en");
}

/**
 * getInternationalDayAbbreviation(), Methode
 *
 * Gibt die lokalisierte 3-Buchst.-Abkürzung des Wochentags zurück
 *
 * @param lang  ISO-Sprachkürzel wie en, de usw.
 */
public function getInternationalDayAbbreviation
                (lang:String):String {
   return this.getInternationalDayName(lang).substring(0, 3);
}

/**
 * getDayAbbreviation(), Methode
 *
 * Gibt die englische 3-Buchst.-Abkürzung des Wochentags zurück
 */
public function getDayAbbreviation():String {
   return this.getInternationalDayAbbreviation("en");
}

/**
 * getInternationalMonthName(), Methode
 *
 * Gibt lokalisierten Monatsnamen nach ISO-Sprachkürzel zurück
 *
```

```
 * @param lang  ISO-Sprachkürzel wie en, de usw.
 */
public function getInternationalMonthName(lang:String):String {
   var months = new Array();
   switch (lang) {
   case "de" :
      months.push("Januar", "Februar", "März", "April", "Mai",
                  "Juni", "Juli", "August", "September",
                  "Oktober", "November", "Dezember");
      break;
   case "en" :
   default :
      months.push("January", "February", "March", "April",
                  "May", "June", "July", "August", "September",
                  "October", "November", "December");
   }
   return months[this.getMonth()];
}

/**
 * getMonthName(), Methode
 *
 * Gibt den englischen Monatsnamen zurück
 *
 */
public function getMonthName() {
   return this.getInternationalMonthName("en");
}

/**
 * getInternationalMonthAbbreviation(), Methode
 *
 * Gibt das lokalisierte 3-Buchst.-Kürzel des Monats zurück
 *
 * @param lang  ISO-Sprachkürzel wie en, de usw.
 */
public function getInternationalMonthAbbreviation(lang:String):String {
   return this.getInternationalMonthName(lang).substring(0, 3);
}

/**
 * getDayAbbreviation(), Methode
 *
 * Gibt 3-Buchst.-Kürzel des englischen Monatsnamens zurück
 */
public function getMonthAbbreviation():String {
   return this.getInternationalMonthAbbreviation("en");
}

/**
 * getInternationalFormatPart(), Methode
 *
 * Gibt lokalisiertes strftime()-kompatibles Zeitelement zurück
```

```
 *
 * @param fPart  strftime()-kompatibler Format-Platzhalter
 * Mögliche Werte:
 * %a - Abgekürzter Wochentag (Sun-Mon)
 * %A - Wochentag (Sunday-Monday)
 * %b - Abgekürzter Monatsname (Jan-Dec)
 * %B - Monatsname (January-December)
 * %c - RFC-1123-Datums/Uhrzeit-String
 * %d - Tag des Monats mit führender 0 (01-31)
 * %e - Tag des Monats (1-31)
 * %H - Stunde (00-23)
 * %I - Stunde (00-12)
 * %j - Tag im Jahr (0-365)
 * %m - numerischer Monat (1-12)
 * %M - Minute (00-59)
 * %p - "a.m." oder "p.m."
 * %S - Sekunde (00-59)
 * %u - numerischer Wochentag (1-7 = Sun-Mon)
 * %w - numerischer Wochentag (0-6 = Sun-Mon)
 * %Y - Jahr (z.B. 2005)
 * %% - %-Zeichen
 *
 * @param lang  ISO-Sprachkürzel wie en, de usw.
 */
public function getInternationalFormatPart
        (fPart:String, lang:String):String {
   var out:String;
   switch (fPart) {
   case "%a" :
      out = this.getInternationalDayAbbreviation(lang);
      break;
   case "%A" :
      out = this.getInternationalDayName(lang);
      break;
   case "%b" :
      out = this.getInternationalMonthAbbreviation(lang);
      break;
   case "%B" :
      out = this.getInternationalMonthName(lang);
      break;
   case "%c" :
      out = this.toString();
      break;
   case "%d" :
      var d:Number = this.getDate();
      out = pre0(d);
      break;
   case "%e" :
      out = String (this.getDate());
      break;
   case "%H" :
      var h:Number = this.getHours();
      out = pre0(h);
      break;
```

```
case "%I" :
   var h:Number = this.getHours();
   if (h > 12) {
      h -= 12;
   }
   out = pre0(h);
   break;
case "%j" :
   var t:Number = 0;
   // Längen der Monate
   var mlengths = new Array
          (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
   // Schaltjahr?
   var j:Number = this.getFullYear();
   if (((j % 4 == 0) && (j % 100 != 0)) || (j % 400 == 0)) {
      mlengths[1] = 29;
   }
   // Tage der Vormonate
   for (var i:Number = 0; i < this.getMonth(); i++) {
      t += mlengths[i];
   }
   // + Tage im aktuellen Monat
   t += this.getDate();
   out = String (t);
   break;
case "%m" :
   var m = this.getMonth() + 1;
   out = pre0(m);
   break;
case "%M" :
   var n = this.getMinutes();
   out = pre0(n);
   break;
case "%p" :
   var h = this.getHours();
   if (h < 12) {
      out = "a.m.";
   } else {
      out = "p.m.";
   }
   break;
case "%S" :
   var s = this.getSeconds();
   out = pre0(s);
   break;
case "%u" :
   out = this.getDay() + 1;
   break;
case "%w" :
   out = String (this.getDay());
   break;
case "%Y" :
   out = String (this.getFullYear());
   break;
```

```
   case "%%" :
      out = "%";
      break;
   default :
      out = "";
   }
   return out;
}

/**
 * getFormatPart(), Methode
 *
 * Gibt englisches strftime()-kompatibles Zeitelement zurück
 *
 * @param fPart  strftime()-kompatibler Format-Platzhalter

 */
public function getFormatPart(fPart:String):String {
   return getInternationalFormatPart(fPart, "en");
}

/**
 * getInternationalFormat(), Methode
  *
 * Wandelt alle strftime()-Platzhalter im String um
 *
 * @param f  strftime()-kompatibler Format-String
 *
 * @param lang  ISO-Sprachkürzel en, de usw.
 */
public function getInternationalFormat
                (f:String, lang:String):String {
   var out:String = "";
     // Den String zeichenweise durchsuchen
   for (var i:Number = 0; i < f.length; i++) {
         // Prozentzeichen gefunden?
      if (f.charAt(i) == "%") {
             // Prozentzeichen plus nächstes ersetzen
         out += this.getInternationalFormatPart
                   (f.substring(i, i + 2), lang);
         i++;
      } else {
             // Zeichen einfach hinzufügen, falls es kein % ist
         out += f.charAt(i);
      }
   }
   return out;
}

/**
 * getFormat(), Methode
```

```
     *
    * Wandelt alle strftime()-Platzhalter im String um (Englisch)
    *
    * @param f  strftime()-kompatibler Format-String
    */
   public function getFormat(f:String):String {
      return getInternationalFormat(f, "en");
   }
}
```

Hier ein Beispiel für die Verwendung dieser Klasse (im Praxisbeispiel von Kapitel 10 wird sie noch einmal konkreter eingesetzt):

```
var dt:CustomDate = new CustomDate();
trace (dt.getInternationallFormat
      ("Heute ist %A, der %d. %B %Y, %H:%M Uhr
        und %S Sekunden - der %j. Tag im Jahr", "de"));
```

Eine Ausgabe dieses Codes sieht zum Beispiel so aus:

```
Heute ist Mittwoch, der 30. November 2005, 22:07 Uhr und 16 Sekunden - der 334.
 Tag im Jahr
```

In diesem Kapitel:

- Weitere Filme laden
- Web-Links mit getURL( )
- Medien dynamisch laden
- Variablen laden
- Praxisbeispiel: Dynamischer Katalog und verknüpfte Website

KAPITEL 8

Mit ActionScript auf externe Daten zugreifen

Das Problem zu kennen ist wichtiger, als die Lösung zu finden,
denn die genaue Darstellung des Problems führt automatisch zur richtigen Lösung.

– Albert Einstein

In Kapitel 6 haben Sie die wichtigsten ActionScript-Funktionen kennen gelernt, mit denen Sie die Komponenten innerhalb eines einzelnen Flash-Films ansprechen und manipulieren können. Einen wesentlich höheren Leistungsumfang erlangt ActionScript aber erst dadurch, dass die Sprache mit verschiedenen externen Dateien kommunizieren kann. In diesem Kapitel lernen Sie die wichtigsten dieser Technologien kennen:

- einen anderen Flash-Film laden
- Web-Hyperlinks mit `getURL()` erstellen
- externe Dateien (Bilder und Sounds) importieren
- Variablen aus Textdateien laden

Weitere Filme laden

Mit den ActionScript-Anweisungen `loadMovieNum()` beziehungsweise `loadMovie()` können Sie auf einfache Art und Weise eine andere SWF-Datei laden. Es gibt dafür zwei Möglichkeiten: Zum einen können Sie den aktuellen Film ersetzen; dies entspricht einer Sprunganweisung zum neuen Film. Zum anderen können Sie den neuen Film aber auch zusätzlich laden – entweder auf eine Stufe oberhalb des ursprünglichen Films, oder aber in eine Movieclip-Instanz hinein. `loadMovieNum()` lädt eine SWF-Datei auf die gewünschte Stufe, wobei die Angabe der Stufe 0 den bisherigen Film ersetzt. `loadMovie()` verwendet dagegen die angegebene Movieclip-Instanz als Ziel.



Beachten Sie, dass Sie mit Hilfe der `loadMovie*()`-Funktionen nur SWF-Dateien laden können. Sie müssen sie also zunächst gemäß der Beschreibung in Kapitel 5, *Flash-Filme veröffentlichen*, exportieren – mit FLA-Arbeitsdateien funktioniert es noch nicht einmal zum Ausprobieren.

Den aktuellen Film ersetzen

Die einfachste Möglichkeit ist das Laden eines anderen Films auf Stufe 0. Dies ersetzt den bisherigen Film, so dass nach wie vor nur ein Film im Flash Player abgespielt wird. Somit brauchen Sie sich nicht mit den speziellen Problemen zu beschäftigen, die auf Sie zukommen können, wenn mehrere Filme gleichzeitig vorhanden sind (siehe nächsten Unterabschnitt). Übrigens hat Stufe 0 noch eine Besonderheit: Da diese Stufe den grundlegenden Film enthält, der alle übrigen steuert, werden beim Laden eines Films auf diese Stufe auch alle eventuell auf höhere Stufen geladenen Filme entfernt.



Der Film, der ursprünglich im Flash Player abgespielt wurde, bestimmt immer die Hintergrundfarbe, die Größe und die Abspielrate. Wenn Sie einen anderen Film auf Stufe 0 laden, übernimmt er diese Werte.

Die Syntax zum Laden eines anderen Films auf Stufe 0 ist einfach:

```
loadMovieNum (URL, 0);
```

Die URL wird als String angegeben, deshalb muss sie in Anführungszeichen stehen, falls sie nicht gerade in einer Variablen oder in einem komplexen Ausdruck steht. Es kann eine lokale Datei oder eine externe URL sein. Wenn Sie sich mit URLs, wie sie zum Beispiel auch in HTML-Hyperlinks benutzt werden, noch nicht auskennen, können Sie im Kasten *Über URLs* in Kapitel 5, *Flash-Filme veröffentlichen*, nachschlagen.

Angenommen, zurzeit wird der Film *menu.swf* abgespielt. Er enthält das mittels `stop()` angehaltene Standbild eines Hauptmenüs mit den beiden Schaltflächen *Katalog* und *Kontakt*. Sie sollen den aktuellen Film beim Anklicken durch die Filme *katalog.swf* beziehungsweise *kontakt.swf* ersetzen, die beide im selben Verzeichnis liegen wie *menu.swf*. Zu diesem Zweck werden der Instanz der *Katalog*-Schaltfläche folgende Anweisungen zugewiesen:

```
on (release) {
  loadMovieNum ("katalog.swf", 0);
}
```

Entsprechend erhält die Schaltfläche *Kontakt* diese Anweisungen:

```
on (release) {
  loadMovieNum ("kontakt.swf", 0);
}
```

Die Stufenhierarchie der Flash-Filme

Das Laden anderer SWF-Filme wird komplizierter, aber auch interessanter, wenn Sie höhere Stufen als 0 angeben: In diesem Fall überlagert der neu geladene Film alle Filme, die auf niedrigeren Stufen liegen. Damit verhalten sich die Stufen so ähnlich wie die Ebenen innerhalb eines einzelnen Flash-Films, wobei jede Stufe einen kompletten Film enthält.

Angenommen, Sie möchten den Film *popup.swf* auf Stufe 1 laden. Dazu können Sie folgenden Code benutzen:

```
loadMovieNum ("popup.swf", 1);
```

Der geladene Film wird an der linken oberen Ecke von Stufe 0 ausgerichtet und nicht skaliert – falls sein Inhalt also größer ist als die Bühne des Grundfilms auf Stufe 0, wird er rechts oder unten abgeschnitten.



Ein Film, der auf eine höhere Stufe geladen wird, besitzt keine geschlossene Hintergrundfläche: Überall, wo keine Inhalte vorhanden sind, erscheint er transparent. Beachten Sie außerdem, dass Schaltflächen auf weiter unten liegenden Stufen auch dann noch funktionieren, wenn sie nicht sichtbar sind, das heißt überdeckt werden.

Wenn Sie einen Film auf eine Stufe laden, auf der sich bereits einer befindet, wird er durch den neuen Film ersetzt. Wie Sie bereits erfahren haben, verhält es sich beim Laden auf Stufe 0 anders: Hier werden die Filme *aller* Stufen geschlossen, weil Stufe 0 alle anderen Stufen steuert.

Sie können Filme auf höheren Stufen auch wieder *entladen*, wenn sie nicht mehr benötigt werden. Dazu wird die Funktion `unloadMovieNum(`*`Stufe`*`)` eingesetzt. Wenn Sie zum Beispiel den Film auf Stufe 3 loswerden möchten, können Sie folgende Anweisung verwenden:

```
unloadMovieNum (3);
```



Es ist technisch ohne weiteres möglich, den Film auf Stufe 0 zu entladen! Dann ist allerdings überhaupt kein Film mehr vorhanden, so dass der Flash Player unwiderruflich »verwaist«. Überprüfen Sie also insbesondere Variablen, die Sie als Stufennummer für `unloadMovieNum()` angeben, damit diese beim Aufruf nicht versehentlich den Wert 0 haben.

Das Interessante an den unterschiedlichen Stufen ist, dass sie sich gegenseitig per ActionScript ansprechen können – Sie können einen Film auf einer anderen Stufe genauso steuern wie einen Movieclip innerhalb eines Films. Der Film einer bestimmten Stufe trägt die Bezeichnung `_level`*`N`*, wobei *`N`* für die Nummer der

gewünschten Stufe steht. Falls Sie also etwa den Film auf Stufe 2 anhalten möchten, lautet die entsprechende Anweisung folgendermaßen:

```
_level2.stop();
```

Natürlich kann ein nachträglich auf eine höhere Stufe geladener Film umgekehrt auch die Stufe 0 beeinflussen. Ihre Bezeichnung ist `_level0`. Das folgende Beispiel macht den Film auf Stufe 0 unsichtbar, indem seine Eigenschaft `_visible` auf `false` gesetzt wird:

```
_level0._visible = false;
```



Die Bezeichnung `_root` bezieht sich immer nur auf den Film, in dem sie verwendet wird – selbst dann, wenn dieser Film auf eine höhere Stufe geladen wird. Es ist nicht etwa ein Synonym für `_level0`.

Externe Filme in Movieclips laden

Sie können SWF-Dateien nicht nur auf eine Stufe über dem aktuellen Film laden, sondern auch in eine beliebige Movieclip-Instanz hinein. Der bisherige Inhalt der Instanz wird dadurch unwiederbringlich entfernt. Es gibt zwei verschiedene Schreibweisen für die passende Anweisung: Sie können `loadMovie()` als globale Funktion oder als Methode einer Movieclip-Instanz aufrufen. Wenn Sie die globale Funktion verwenden, müssen Sie zwei Parameter angeben, nämlich die URL der gewünschten SWF-Datei sowie einen Bezug auf die Instanz, in die diese hineingeladen werden soll. Die Syntax der Funktion sieht also folgendermaßen aus:

```
loadMovie (SWF-URL, Instanz);
```

Wenn Sie also über diese Syntax eine Datei namens *test.swf* aus dem aktuellen Verzeichnis in die Movieclip-Instanz `box` im Hauptfilm laden möchten, sieht die entsprechende Anweisung so aus:

```
loadMovie ("test.swf", _root.box);
```

Die Methoden-Syntax ist praktischer:

```
Instanz.loadMovie (SWF-URL);
```

In dieser Schreibweise sieht das obige Beispiel folgendermaßen aus:

```
_root.box.loadMovie ("test.swf");
```

Die Methode ist übrigens auch sicherer als die globale Funktion: Wenn die angesprochene Movieclip-Instanz nicht existieren sollte, passiert nichts weiter. Bei der Funktion wird dagegen der *aktuelle* Movieclip (im ungünstigsten Fall also der Hauptfilm!) ersetzt.

Die linke obere Ecke des nachgeladenen Films befindet sich übrigens auf dem Registrierpunkt der Movieclip-Instanz – falls Sie die ganze Bühne exakt abdecken möchten, müssen Sie den Clip also so platzieren, dass sein Registrierpunkt auf der linken oberen Bühnenecke liegt.

Wenn Sie einen Film in eine Movieclip-Instanz hineinladen, verhält er sich ein wenig anders als auf einer eigenen Stufe: Zum einen kann er sich *hinter* eingebauten Inhalten des Films befinden, in den er geladen wird – sogar hinter einer Maske. Zum anderen kann er beliebig transformiert (skaliert, gedreht oder gespiegelt) werden: Wenn Sie die zugrunde liegende Instanz zuvor auf der Bühne transformiert haben, gilt dies auch für den hineingeladenen Film. Abgesehen davon kann die Instanz auch per Bewegungs-Tweening skaliert werden, so dass ein neu geladener Film in eine beliebige Bewegung versetzt oder sogar mit Hilfe eines Farbeffekts eingefärbt werden kann.



In einem Film, den Sie in eine Instanz hineinladen, müssen Sie mit ActionScript-Anweisungen sehr vorsichtig sein: `_root` bezieht sich nicht mehr auf den Film, in dem die Anweisung ursprünglich stand, sondern auf den Hauptfilm, in den dieser geladen wurde. Auch die anderen Bezüge ändern sich entsprechend.

Wenn Sie dies verhindern möchten, können Sie seit Flash MX 2004 die Eigenschaft `_lockroot` des entsprechenden Movieclips auf `true` setzen – dies sorgt dafür, dass `_root` *nicht* neu interpretiert wird.

Auch Filme, die Sie in Movieclip-Instanzen hineingeladen haben, können Sie bei Bedarf wieder entfernen. Dazu wird die Funktion oder Methode `unloadMovie()` verwendet. Als globale Funktion besitzt die Anweisung die folgende Syntax:

```
unloadMovie (Instanz);
```

Die objektorientierte Variante sieht dagegen so aus:

```
Instanz.unloadMovie();
```

Um die zuvor in die Instanz `_root.box` geladene Instanz wieder zu entfernen, können Sie eine der beiden folgenden Anweisungen verwenden:

```
unloadMovie (_root.box);   // globale Funktion
_root.box.unloadMovie();   // Movieclip-Methode
```

Filme vorausladen

Normalerweise arbeitet Flash im Streaming-Betrieb: Sobald genügend Daten heruntergeladen wurden, um das nächste Frame anzuzeigen, wird es abgespielt. In den meisten Fällen ist das auch in Ordnung; es ist sogar ein besonderer Vorteil gegenüber Formaten, die vor dem Abspielen stets komplett heruntergeladen werden müssen. In Einzelfällen kann es allerdings vorkommen, dass Inhalte zu datenintensiv für eine durchschnittliche Netzwerkbandbreite sind. Wenn Sie dies – zum Beispiel über die Mittel, die in Kapitel 5 vorgestellt wurden – festgestellt haben, sollten Sie den entsprechenden Teil des Films vorausladen.

Sowohl der Hauptfilm als auch jeder Movieclip verfügen über die Eigenschaften `_framesloaded` und `_totalframes`, die die Anzahl der bereits geladenen beziehungsweise die Gesamtzahl der Bilder angeben. Alternativ lassen sich die Methoden `getBytesLoaded()` und `getBytesTotal()` einsetzen, die dieselbe Aufgabe für die Bytes eines Films oder Movieclips erledigen.

Angenommen, Sie möchten warten, bis der gesamte Film geladen ist. Fügen Sie dazu vor der bisherigen Anfangsszene eine neue Szene namens `loader` ein. Weisen Sie Bild 2 dieser Szene folgendes Bildskript zu:

```
stop();
```

Setzen Sie zuletzt einen Steuerclip (Instanz eines leeren Movieclips) auf die Bühne. Versehen Sie ihn mit folgenden ActionScript-Anweisungen:

```
onClipEvent (enterFrame) {
   if (_root._framesloaded == _root._totalframes) {
      _root.play();
   }
}
```

Natürlich können Sie statt eines leeren Movieclips auch ein Symbol mit einer kleinen »wird geladen«-Animation oder Ähnliches verwenden, um die Wartezeit unterhaltsamer zu gestalten. Sehr beliebt sind auch Fortschrittsbalken. Wenn der Movieclip gleichzeitig den Ladefortschritt anzeigen soll, benötigt er als Inhalt einen waagerechten Balken, der sich vom Registrierpunkt aus nach rechts fortsetzt. Diese Platzierung sorgt dafür, dass er in die richtige Richtung wächst. Tauschen Sie das obige Skript zum Schluss gegen das folgende aus:

```
onClipEvent (enterFrame) {
   if (_root._framesloaded == _root._totalframes) {
      _root.play();
   } else {
      this._xscale = _root.getBytesLoaded() / _root.getBytesTotal() * 100;
   }
}
```

Web-Links mit getURL( )

Wenn eine SWF-Datei in einem Browser angezeigt wird, kann sie genau wie ein HTML-Dokument dafür sorgen, dass der Browser ein anderes Dokument anfordert. Dafür ist die Funktion `getURL()` zuständig. Übrigens können dynamische Textfelder sogar reguläre HTML-Hyperlinks enthalten – dies wird im nächsten Kapitel angesprochen. `getURL()` benötigt ein bis drei Argumente; die allgemeine Syntax sieht so aus:

```
getURL (URL [, Ziel [, Methode]]);
```

Die *`URL`* ist zwingend erforderlich; es handelt sich natürlich um die Adresse der Ressource, die der Browser anfordern soll. Das *`Ziel`* legt fest, in welches Browserfenster

oder Frame das neue Dokument geladen werden soll. Die *Methode* ("GET" oder "POST") brauchen Sie schließlich nur zu verwenden, wenn Ihr Flash-Film seine Variablen an die URL senden soll. Dies ergibt nur dann einen Sinn, wenn ein serverseitiges Skript angesprochen wird (siehe Kapitel 10, *Interaktion mit Webserver-Anwendungen*).

Wenn Sie das folgende Skript einer Schaltfläche zuweisen, wird bei einem Klick im aktuellen Browserfenster die Homepage des O'Reilly-Verlags geöffnet:

```
on (release) {
   getURL ("http://www.oreilly.de/");
}
```

Mit Fenstern und Frames arbeiten

In diesem Unterabschnitt soll der zweite Parameter von `getURL()`, das *Ziel*, näher beleuchtet werden. Wie bereits erwähnt, handelt es sich um die Angabe des Browserfensters oder Frames, in das die neue URL geladen werden soll. Falls es sich um ein Frame handelt, wird dessen selbst definierter Name angegeben. Daneben gibt es vier vordefinierte Namen, die jeder Browser versteht:

- `_self` – das aktuelle Frame beziehungsweise Fenster; dies ist Standard
- `_parent` – das übergeordnete Frame
- `_top` – das gesamte Browserfenster, egal wie tief das aktuelle Frame verschachtelt ist
- `_blank` – ein neues, leeres Browserfenster

Die folgende Anweisung lädt die Datei *seite2.html* aus dem aktuellen Verzeichnis in das Frame *inhalt*:

```
getURL ("seite2.html", "inhalt");
```

Mit dieser Anweisung wird die Website von Macromedia in das gesamte Browserfenster geladen:

```
getURL ("http://www.macromedia.com/", "_top");
```

Die folgende Anweisung öffnet schließlich ein neues Fenster und zeigt darin die Datei *start.html* aus dem übergeordneten Verzeichnis an:

```
getURL ("../start.html", "_blank");
```

Spezielle URLs

Übrigens können Sie mit `getURL()` nicht nur URLs angeben, die neue Dokumente laden. Es gibt einige spezielle URLs für Sonderaufgaben. Zwei von ihnen – E-Mail- und JavaScript-URLs – werden hier kurz vorgestellt.

Es ist überaus praktisch, wenn man auf einer Webseite einen Link anklicken kann, um jemandem eine E-Mail zu senden. Zahlreiche Sites enthalten auf jeder einzelnen Seite einen allgemeinen E-Mail-Link und sind zudem oftmals mit einer Kontaktseite

Frames

Frames sind eine Möglichkeit, mehrere Webseiten in einer bestimmten Anordnung in ein und demselben Browserfenster anzuzeigen. Sie wurden von Netscape entwickelt und zuerst in den Netscape Navigator 2.0 eingebaut. Anfangs erfreuten sie sich unter Webdesignern großer Beliebtheit. Schließlich bieten sie unter anderem den Vorteil, dass nicht bei jedem Klick auf einen Link die gesamte Seite neu geladen werden muss. Gerade bei Internetverbindungen mit geringer Bandbreite ist dies ein großer Vorteil, so dass um 1998 fast jede Site aus einem Frame mit einem feststehenden Navigationsbereich und einem weiteren mit wechselnden Inhalten bestand. Erst mit steigenden Geschwindigkeiten traten die Nachteile der Frames in den Vordergrund: Frame-basierte Seiten lassen sich nicht ordentlich in der Favoriten-Liste eines Browsers speichern. Zudem ist es problematisch, wenn Suchmaschinen nur den Inhalt einzelner Frames finden. Aus diesen Gründen sollten neu konzipierte Sites in aller Regel auf den Einsatz von Frames verzichten.

Der HTML-Code zur Definition von Frames ist nicht weiter kompliziert: Eine erste HTML-Seite teilt das Fenster horizontal und/oder vertikal in mehrere Frames auf und lädt anschließend andere Dokumente hinein. Das Frame-Definitions-Dokument enthält statt eines `<body>`-Tags ein `<frameset>`-Tag. Das folgende Beispiel unterteilt das Fenster in einen linken Bereich von 120 Pixeln Breite und einen beliebig breiten rechten Bereich:

```
<html>
  <head>
    <title>Frameset</title>
  </head>
  <frameset cols="120, *">
    <frame src="navi.html" />
    <frame src="start.html" name="inhalt" />
  </frameset>
</html>
```

Der Parameter `name` im zweiten (rechten) Frame besagt, dass Hyperlinks aus einem anderen Frame heraus (zum Beispiel aus dem linken) das Ziel `"inhalt"` angeben können, um ein neues Dokument in dieses Frame zu laden. Ein HTML-Hyperlink in der Datei *navi.html*, der die Seite *kontakt.html* in das Frame *inhalt* laden soll, sieht zum Beispiel so aus:

```
<a href="kontakt.html" target="inhalt">Kontakt</a>
```

Falls die Datei *navi.html* dagegen einen Flash-Film mit Schaltflächen zur Navigation enthielte, dann würde der Schaltfläche *Kontakt* das folgende Skript zugewiesen:

```
on (release) {
   getURL ("kontakt.html", "inhalt");
}
```

ausgestattet, die die persönlichen E-Mail-Adressen bestimmter Ansprechpartner angibt. Wenn ein Benutzer im Browser auf einen Mail-Link klickt, öffnet sich normalerweise das bevorzugte E-Mail-Programm des Benutzers; die Empfängeradresse ist bereits eingetragen. Dies funktioniert mit einer URL nach dem Schema *mailto: E-Mail-Adresse*. Das folgende Beispiel kann einer Schaltfläche zugewiesen werden und öffnet dann auf Knopfdruck ein Mail-Fenster mit einer neuen Nachricht an *info@flashrock-music.com*:

```
on (release) {
   getURL ("mailto:info@flashrock-music.com");
}
```

Optional besteht die Möglichkeit, auch den Betreff der E-Mail bereits festzulegen. Zu diesem Zweck müssen Sie *?subject=Betreffzeile* an die URL anhängen. Die folgende Anweisung öffnet ein Mail-Fenster mit einer Mail an *info@flashrock-music.com* und dem Betreff »Bestellung«:

```
getURL ("mailto:info@flashrock-music.com?subject=Bestellung");
```

Eine weitere nützliche Art von URLs besitzt das Schema *javascript:*. Es ermöglicht den Aufruf einer beliebigen JavaScript-Anweisung – sowohl einer vorgefertigten als auch einer Funktion, die im aktuellen HTML-Dokument definiert ist. Wenn Sie `getURL()` mit einer solchen URL benutzen, können Ihre Flash-Filme mit JavaScript-Anwendungen kommunizieren. Das folgende Beispiel weist den Browser an, ein kleines Alarmfenster mit der Meldung `"Hallo, hier Flash!"` anzuzeigen:

```
getURL ("javascript:alert (\"Hallo, hier Flash!\");");
```

Wie der Aufruf der Methode `alert()` zeigt, ist die JavaScript-Grundsyntax mit derjenigen von ActionScript identisch. Auch alle anderen Sprachgrundlagen – Variablen, Funktionen, Kontrollstrukturen und so weiter – funktionieren genau gleich.

Das nächste Beispiel ruft eine selbst definierte Funktion auf. Fügen Sie zunächst über dem Abschluss-Tag `</head>` des HTML-Dokuments, in das Sie den aufrufenden Flash-Film hineinladen, diese Zeilen ein:

```
<script language="JavaScript" type="text/javascript">

   function popupFenster ()
   {
      // Werbe-Pop-up öffnen
      var win = open ("werbung.html", "",
      "toolbars=0,menubar=0,location=0,directories=0,status=0,width=480,height=360");
      // Bildschirm messen
      var sx = screen.width;
      var sy = screen.height;
      // Pop-up in der Mitte platzieren
      win.moveTo ((sx - 480) / 2, (sy - 360) / 2);
   }

</script>
```

Die JavaScript-Methode `open (URL, Ziel-Name, Feature-Liste)` öffnet ein neues Browserfenster, in das die angegebene URL geladen wird; die Feature-Liste bestimmt die Ausstattung und die Größe des Fensters. Die Methode `moveTo()` verschiebt das Fenster anschließend genau in die Bildschirmmitte. Fügen Sie nun folgende Anweisung in ein Bild-Skript des Flash-Films ein, um diese Funktion aufzurufen:

```
getURL ("javascript:popupFenster();");
```

Flash-Filme per JavaScript steuern



Umgekehrt besteht übrigens auch die Möglichkeit, das Verhalten von Flash-Filmen durch JavaScript-Anweisungen zu steuern. Der Internet Explorer verwendet dafür einige Methoden, die dem eingebetteten ActiveX-Objekt zugeordnet sind. Netscape-kompatible Browser verwenden dagegen eine Technologie namens LiveConnect, die nicht ganz so viele Methoden unterstützt.

Wenn Sie einen eingebetteten SWF-Film im Internet Explorer per JavaScript steuern möchten, benötigt dieser eine eindeutige ID, die über das gleichnamige Attribut im `<object>`-Tag festgelegt wird. Erfreulicherweise erledigt die Funktion *Veröffentlichen* dies automatisch. Beispiel:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.
cab#version=8,0,0,0" width="550" height="400" id="jssteuerung" align="middle"
> ... </object>
```

Der Flash-Film lässt sich hier über das JavaScript-Konstrukt `window.jssteuerung` ansprechen.

Die anderen Browser brauchen dagegen das Attribut `name` im `<embed>`-Tag, das normalerweise ebenfalls durch Veröffentlichen eingetragen wird. Vorsichtshalber sollten Sie – für ältere Browser – noch das Attribut `swliveconnect="true"` hinzufügen:

```
<embed src="jssteuerung.
swf" quality="high" bgcolor="#ffffff" width="550" height="400"
name="jssteuerung" swliveconnect="true" align="middle"
allowScriptAccess="sameDomain" type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer"></embed>
```

Nun lässt sich der Film als `document.embeds['jssteuerung']` oder `document.embeds.jssteuerung` ansprechen.

Wenn Sie die in Kapitel 5 vorgestellte und beim Veröffentlichen automatisch erstellte Verschachtelung von `<object>` und `<embed>` einsetzen, sollten Sie eine *Browserweiche* benutzen, um den JavaScript-Code in beiden Varianten hinzufügen zu können:

→

```
// 'MSIE' in Browser-Beschreibung?
if (navigator.userAgent.indexOf ('MSIE') >= 0) {
   // Ja - MS Internet Explorer
   // ... MSIE-Code
} else {
   // Nein - Netscape
   // ... Netscape-Code
}
```

Hier die wichtigsten JavaScript-Methoden, die Sie in beiden Browser-Sorten aufrufen können (beachten Sie die Groß- und Kleinschreibung!):

- `Play()` – Abspielen
- `StopPlay()` – Anhalten
- `GotoFrame(`*`Nummer`*`)` – entspricht der ActionScript-Funktion `gotoAndStop()`
- `SetVariable (`*`Var, Wert`*`)` – setzt die Variable *`Var`* im Film auf den angegebenen *`Wert`*
- `GetVariable (`*`Var`*`)` – liest den Wert der angegebenen ActionScript-Variablen aus

Das folgende Beispiel stoppt den Film `jssteuerung` im Internet Explorer:

```
window.jssteuerung.StopPlay();
```

Diese Anweisung schickt dieselbe SWF-Datei in Netscape zu Frame Nummer 20:

```
document.embeds['jssteuerung'].GotoFrame (20);
```

Medien dynamisch laden

In Kapitel 4 haben Sie erfahren, wie Sie Mediendateien manuell in die Arbeitsumgebung von Flash importieren und sinnvoll einsetzen können. Alternativ können Sie *JPEG-Bilder* und *MP3-Sound-Dateien* auch dynamisch per ActionScript laden.

JPEG-Bilder laden

Das Praktische an den Funktionen zum Laden von JPEG-Bildern ist, dass Sie diese bereits kennen: `loadMovieNum()` beziehungsweise `loadMovie()` können nicht nur SWF-Dateien, sondern eben auch JPEGs laden.

Das folgende Beispiel lädt die Datei *foto.jpg* aus dem aktuellen Ordner auf Stufe 1:

```
loadMovieNum ("foto.jpg", 1);
```

Nützlicher dürfte es in der Regel sein, JPEG-Bilder in Movieclip-Instanzen zu laden. Auf diese Weise lässt sich nämlich genau festlegen, wo sie auf der Bühne platziert werden sollen: mit ihrer linken, oberen Ecke am Registrierpunkt der Instanz. Das folgende Beispiel lädt die Datei *gitarre.jpg* aus dem untergeordneten Verzeichnis *bilder* in den Movieclip `bildbox`:

```
bildbox.loadMovie ("gitarre.jpg");
```

Selbstverständlich ist auch hier wieder die alternative Schreibweise von `loadMovie()` als globale Funktion zulässig:

```
loadMovie ("gitarre.jpg", bildbox);
```

MP3-Dateien laden

Wenn Sie MP3-Sounds dynamisch laden möchten, müssen Sie eine Instanz der Klasse `Sound` verwenden. Diese wird wie folgt erstellt:

```
var klang = new Sound();
```

Anschließend können Sie folgende Methode des Objekts benutzen, um eine Sound-Datei zu laden:

```
klang.loadSound (URL, Stream);
```

Die *URL* gibt die Quelle der Sounddatei an, während der `Boolean`-Parameter *Stream* den Wert `true` für *Stream-* oder `false` für *Ereignis-Sound* erhält. Das folgende Beispiel lädt die lokale Datei *knall.mp3* als Ereignis-Sound:

```
klang.loadSound ("knall.mp3", false);
```

Der nachfolgende Code lädt dagegen die Datei *track2.mp3* aus dem untergeordneten Ordner *musik* als Stream-Sound:

```
klang.loadSound ("musik/track2.mp3", true);
```

Sound-Objekte ermöglichen übrigens auch die dynamische Steuerung des Sounds, der darin abgespielt wird. Beispielsweise regelt ihre Methode `setVolume()` die Gesamtlautstärke; die zulässigen Werte liegen zwischen 0 (aus) und 100 (Maximum; dies ist die Voreinstellung). Die folgende Anweisung setzt die Lautstärke von `klang` auf 50%:

```
klang.setVolume(50);
```

Die aktuelle Lautstärke können Sie übrigens mittels `klang.getVolume()` auslesen.

Mit Hilfe der Methoden `setPan()` und `getPan()` können Sie auf ähnliche Weise die Balance setzen beziehungsweise auslesen. Die Werte reichen von -100 (nur linker Kanal) über 0 (neutral) bis +100 (nur rechter Kanal).

Variablen laden

Eine letzte bemerkenswerte Fähigkeit von SWF-Filmen besteht darin, dass sie Variablen aus externen Textdateien laden können. Besonders sinnvoll wird dies im Zusammenhang mit serverseitigen Skripten, die diese Variablen dynamisch generieren. Dieses Thema wird ausführlich in Kapitel 10 besprochen.

Damit eine Textdatei als Quelle für Flash-Variablen in Frage kommt, müssen diese in der Form `Var1=Wert1&Var2=Wert2...` drinstehen. Zu allem Überfluss müssen zahl-

reiche Sonderzeichen in einer solchen Datei speziell kodiert werden (*URL-Kodierung*): Aus dem Leerzeichen wird ein +; die meisten Satzzeichen oder Umlaute werden durch ein Prozentzeichen und ihren hexadezimalen Zeichencode angegeben: Beispielsweise steht `%FC` für *ü*, während `%22` das Anführungszeichen repräsentiert.[1]

Speichern Sie als Beispiel den folgenden Textblock unter dem Namen *vars.txt*:

```
hersteller=Fender&modell=Stratocaster&baujahr=1980&farbe=rot
```

Weisen Sie einem Bild in einem Flash-Film folgendes Skript zu:

```
loadVariablesNum ("vars.txt", 0);
trace ("Hersteller: " + hersteller);
trace ("Modell: " + modell);
trace ("Baujahr: " + baujahr);
trace ("Farbe: " + farbe);
```

Die Funktion `loadVariablesNum()` besitzt folgende allgemeine Syntax:

```
loadVariablesNum (URL, Stufe[, Methode]);
```

Auch hier können Sie also optional `"GET"` oder `"POST"` angeben, um zunächst die vorhandenen Variablen an die angesprochene URL zu senden. In den nächsten beiden Kapiteln erfahren Sie, wie sich – unter anderem mit dieser Funktion – Formulare erstellen lassen. Das zweite Argument ist die Stufe, wie Sie sie von `loadMovieNum()` her kennen.

`loadVariables()` lädt die Variablen aus der angegebenen Datei nicht in den Hauptfilm oder auf eine höhere Stufe, sondern in eine Movieclip-Instanz. Genau wie `loadMovie()` können Sie auch diese Anweisung sowohl als normale Funktion als auch als Movieclip-Instanzaktion schreiben. Jedes der beiden folgenden Beispiele lädt die Variablen aus der Datei *vars2.txt* in den Movieclip `varclip`:

```
loadVariables ("vars2.txt", varclip);     // klassisch
varclip.loadVariables ("vars2.txt");      // objektorientiert
```

Praxisbeispiel: Dynamischer Katalog und verknüpfte Website

Das dynamische Laden von JPEG-Bildern sowie von Variablen wird hier an einer neuen Variante eines Produktkatalogs gezeigt, in dem die Abbildungen und ihre Beschreibungen beziehungsweise Werbetexte aus externen Dateien stammen. Anschließend werden sämtliche bisher vorgestellten Dokumente zu einer kompletten Website verbunden.

---

1 Einfache Textdateien geben keinen Zeichensatz vor, deshalb klappt es beim Laden normalerweise nicht mit Umlauten.

Produktkatalog mit dynamischen Bildern

Wenn Sie eine Site mit zahlreichen Abbildungen betreiben, bietet es sich an, diese Bilder dynamisch zu laden. Am besten nummerieren Sie die Dateinamen nach einem einfachen Schema durch, um auch die Ladebefehle selbst zu automatisieren. Ein realer Online-Shop ist sicherlich ein gutes Beispiel für eine solche bilderreiche Website. Das vorliegende Beispiel ist deshalb ein weiterer Abbildungskatalog, wie er bereits in den Praxisteilen der Kapitel 4 und 6 vorgestellt wurde, diesmal allerdings mit dynamisch geladenen JPEG-Bildern.

Öffnen Sie die Datei *dynloader_start.fla* aus dem CD-Verzeichnis *beispiele/kapitel8*. Auf der Bühne finden Sie lediglich das Logo, ein dynamisches Textfeld für die Beschreibungen sowie zwei Schaltflächeninstanzen zum Blättern. Die JPEG-Dateien *artikel1.jpg* bis *artikel13.jpg* sollen nicht in die Flash-Arbeitsumgebung importiert werden; diese Aufgabe soll ActionScript erledigen. Auch die Beschreibungsdateien *artikel1.txt* bis *artikel13.txt* werden dynamisch geladen.

Klicken Sie das Textfeld an, und ordnen Sie ihm unter *Var* in der Eigenschaftenleiste den Variablennamen `beschreibung` zu.

Erstellen Sie ein neues Movieclip-Symbol mit der Bezeichnung *bilderrahmen*. Fügen Sie keine Inhalte ein, sondern ziehen Sie eine Instanz dieses leeren Symbols auf die Bühne. Sie soll rechts oben neben dem Logo platziert werden. Anders als die in Kapitel 6 verwendeten leeren Movieclips wird dies aber kein Steuerclip, sondern die Instanz, in die die Bilder hineingeladen werden sollen. Weisen Sie ihr den Instanznamen *rahmen* zu.

Erstellen Sie nun in Bild 2 der Ebene *actions* folgendes Skript:

```
rahmen.loadMovie ("artikel1.jpg");
loadVariablesNum ("artikel1.txt", 0);
var artikelNr = 1;
stop();
```

Dies lädt zu Anfang das Bild *artikel1.jpg* in die Instanz *rahmen* und setzt die »Blätter-Variable« `artikelNr` auf den zugehörigen Startwert 1. `stop()` sorgt wie üblich dafür, dass der Film an dieser Stelle anhält.

Der linken Schaltfläche (Instanz von *back*) werden diese Anweisungen zugewiesen:

```
on (release) {
  artikelNr--;
  if (artikelNr < 1) {
      artikelNr = 13;
  }
  rahmen.loadMovie ("artikel" + artikelNr + ".jpg");
  loadVariablesNum ("artikel" + artikelNr + ".txt", 0);
}
```

Das Skript der anderen Schaltfläche sieht nahezu identisch aus:

```
on (release) {
  artikelNr++;
  if (artikelNr > 13) {
      artikelNr = 1;
  }
  rahmen.loadMovie ("artikel" + artikelNr + ".jpg");
  loadVariablesNum ("artikel" + artikelNr + ".txt", 0);
}
```

Damit dieses Beispiel funktioniert, müssen Sie die SWF-Datei des Films in ein Verzeichnis exportieren, in dem sich auch die JPEG-Bilddateien und die Textdateien befinden. Alternativ können Sie auch den passenden Pfad beziehungsweise die URL des Bilder-Verzeichnisses angeben.

Die Website zusammenstellen

Größere Flash-basierte Websites bestehen oftmals aus vielen einzelnen Flash-Filmen, die einander über `loadMovieNum()` oder gar durch Austausch der HTML-Dokumente mittels `getURL()` aufrufen. Die Website des FlashRock Music Shop ist ein ideales Beispiel dafür. Deshalb eignet sich dieses Kapitel hervorragend dafür, aus den zahlreichen Einzeldateien der bisherigen Kapitel eine vollständige Website zu machen, an der einige der hier vorgestellten Techniken demonstriert werden.

Im Verzeichnis *beispiele/kapitel8/site_start* finden Sie die Ausgangsdateien der Site im FLA-Format. Die meisten von ihnen haben Sie unter anderem Namen bereits selbst erstellt. Hier wurden sie lediglich um einen Button für die Rückkehr zum Hauptmenü erweitert. Die Datei, um deren Bearbeitung es hier vor allem geht, bildet das Hauptmenü. Sie finden sie unter dem vorläufigen Namen *menu_start.fla*.

Die Datei enthält in insgesamt fünf Bildern zahlreiche Schaltflächen, die dazu dienen sollen, andere Filme beziehungsweise externe URLs zu laden. Als Erstes werden die Schaltflächen im Bild mit der Markierung *main* bearbeitet. Sie erhalten lediglich Sprungbefehle zu den anderen Frames desselben Films, weil sie die übergeordnete Ebene des Hauptmenüs bilden. Klicken Sie die Schaltfläche *Produkte* an, und weisen Sie ihr folgendes Skript zu, das zum Bild mit der Markierung *prod* springt:

```
on (release) {
   gotoAndStop ("prod");
}
```

Die Schaltfläche *Info* erhält die folgenden Anweisungen, die beim Anklicken einen Sprung zum Frame *info* durchführen:

```
on (release) {
   gotoAndStop ("info");
}
```

Auch *Kontakt* wird mit einem ähnlichen Skript ausgestattet; es verzweigt zum Bild *kont*:

```
on (release) {
   gotoAndStop ("kont");
}
```

Als Letztes bekommt nun der Button *Links* seine ActionScript-Befehle; sie sorgen für einen Sprung zum Frame *link*:

```
on (release) {
   gotoAndStop ("link");
}
```

Wechseln Sie nun in das Bild *prod*. Hier finden Sie drei Schaltflächen, die mit Skripten ausgestattet werden sollen. Weisen Sie zunächst dem Button *Katalog* das folgende Skript zu:

```
on (release) {
   loadMovieNum ("katalog.swf", 0);
}
```

Die Schaltfläche *Bestellen* bekommt folgende Anweisungen:

```
on (release) {
   loadMovieNum ("order.swf", 0);
}
```

Schließlich wird noch der Button *Hauptmenü* mit Code versehen:

```
on (release) {
   gotoAndStop ("main");
}
```

Im Frame *info* finden Sie vier Schaltflächen. Sorgen Sie nach dem bewährten Schema dafür, dass der Button *Gitarrenkunde* die Datei *aufbau.swf* auf Stufe 0 lädt. Ebenso soll *Akkorde* den Film *akkorde.swf* laden und die Schaltfläche *Unterhaltung* den Film *spiel.swf*. Der Button *Hauptmenü* soll wieder zum Frame *main* zurückkehren.

Im Bild *kont* finden Sie nur zwei Schaltflächen. Sorgen Sie dafür, dass *Gästebuch* den Film *guest.swf* lädt, während *Hauptmenü* wie üblich zum Frame *main* springen soll.

Wechseln Sie in das Bild *link*. Hier befindet sich eine kleine Link-Liste. Weisen Sie der Schaltfläche *Fender* folgendes Skript zu:

```
on (release) {
   getURL ("http://www.fender.com/", "_blank");
}
```

Statten Sie die restlichen Schaltflächen entsprechend mit Skripten aus, damit sie folgende Sites jeweils in einem neuen Fenster (`"_blank"`) öffnen:

- Gibson: *http://www.gibson.com/*
- Washburn: *http://www.washburn.com/*
- Marshall: *http://www.marshallamps.com/*

- Jim Dunlop: *http://www.jimdunlop.com/*
- Red Hot Chili Peppers: *http://www.redhotchilipeppers.com/*
- Nuno Bettencourt: *http://www.nuno-bettencourt.com/*
- Remedy Inc.: *http://www.remedy-inc.com/*
- Gitarre & Bass: *http://www.gitarrebass.de/*

Zu guter Letzt soll der Button *Hauptmenü* auch hier zum Frame *main* zurückkehren.

Speichern Sie die Datei nun unter dem Namen *menu.fla*, und schließen Sie sie zunächst. Öffnen Sie nun die Datei *intro.fla*. Stellen Sie unter *Datei → Einstellungen für Veröffentlichungen* gegebenenfalls die Standardoptionen (SWF und HTML) wieder her. Veröffentlichen Sie die Datei, und benennen Sie das neu entstandene HTML-Dokument *intro.html* in *index.html* um.

Öffnen Sie nun alle anderen FLA-Dateien aus dem Arbeitsordner. Exportieren Sie jede einzelne von ihnen als SWF-Datei (siehe Kapitel 5). Im endgültigen Ordner, in dem die Dateien liegen sollen, können Sie die Original-FLA-Dateien nun entfernen.



Damit die Site vollständig funktioniert, muss sie in einem Webserver-Verzeichnis liegen, in dem PHP aktiviert ist, und anschließend per `http:` geladen werden. Einzelheiten dazu finden Sie in Kapitel 10.

In diesem Kapitel:

- Webformulare im Überblick
- Textfelder und Komponenten in Flash
- Praxisbeispiel: Ein Bestellformular

KAPITEL 9

Formulare erstellen

Ausführungsbestimmungen sind Erklärungen zu den Erklärungen,
mit denen man eine Erklärung erklärt.

– Abraham Lincoln

Webformulare im Überblick

Sobald Sie das World Wide Web nutzen, haben Sie ständig mit Webformularen zu tun. Sie dienen der Kontaktaufnahme, der Bestellung von Waren und Dienstleistungen oder stellen die Web-Schnittstelle zu beliebigen Anwendungen zur Verfügung. Solche Formulare lassen sich nicht nur in HTML realisieren, sondern sind auch in Flash nicht schwer umzusetzen. Letztendlich sind Formulare nämlich nichts weiter als Hyperlinks, die zusätzlich die eingegebenen Daten an die angeforderte Adresse schicken. Unter dieser URL ist dann aber in aller Regel kein statisches Dokument zu finden, sondern ein serverseitiges Programm oder Skript, das die Daten in Empfang nimmt und auswertet.

Diese »Rückseite« der Formulare wird im nächsten Kapitel vorgestellt; hier geht es zunächst um die Formulare selbst: Sie lernen zunächst einige Hintergründe des Web-Protokolls HTTP kennen, das die Formulardaten transportiert. Anschließend erhalten Sie einen kurzen Einblick in den Aufbau klassischer HTML-Formulare. Den Hauptteil dieses Kapitels bildet die Beschreibung von dynamischen Textfeldern und Komponenten, die sich übrigens nicht nur für den externen Formularversand eignen.

Die HTTP-Methoden GET und POST

`GET` und `POST` sind zwei verschiedene Arten von HTTP-Anfragen, die ein Client (beispielsweise ein Browser oder der Flash Player) an einen Webserver senden. Beide Arten von Anfragen können Formulardaten transportieren, unterscheiden sich aber ein wenig voneinander.

Um den genauen Unterschied zu verstehen, sollten Sie sich zunächst einmal vor Augen führen, wie eine HTTP-Transaktion überhaupt funktioniert. Als Beispiel soll eine Browser-Anfrage für die O'Reilly-Seite zu diesem Buch (*http://www.oreilly.de/catalog/flashmxbas2ger/*) dienen. Diese Adresse wird zunächst in den Browser eingegeben; meist ohne das vorangestellte *http://*, da es Standard ist. Der Browser trennt die Bestandteile der URL voneinander: Anfrageschema *http*, angesprochener Rechner *www.oreilly.de* und Pfad */catalog/flashmxbas2ger/*. Als Nächstes ermittelt der Browser über das Domain Name System (DNS) die echte Adresse (IP-Adresse) hinter *www.oreilly.de* (zurzeit 62.206.71.33), baut eine Netzwerkverbindung zu dieser Adresse auf und sendet an sie die folgende HTTP-Anfrage:

```
GET /catalog/flashmxbas2ger/ HTTP/1.1
Accept: */*
Accept-Language: de, en
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.01)
Host: www.oreilly.de
Connection: Keep-Alive
```

Relevant ist vor allem die erste Zeile der Anfrage: `GET` ist die bereits erwähnte Anfragemethode. Sie dient im Unterschied zu anderen Methoden der einfachen Anforderung einer Ressource, die der Client anzeigen oder herunterladen möchte. Die zweite Komponente ist der Pfad der Ressource, zum Schluss folgt die Angabe der HTTP-Protokollversion (1.1 ist aktuell). Die restlichen Zeilen sind zusätzliche *Header*. Sie sind zwar hier nicht weiter wichtig, aber für die Interessierten unter Ihnen hier kurz ihre Bedeutung:

- `Accept: */*`: Zeigt an, dass der Browser bereit ist, alle Arten von Dateien anzunehmen.
- `Accept-Language: de, en`: Der Browser bevorzugt Dokumente in deutscher und englischer Sprache, in dieser Reihenfolge. Manche Server bieten Dateien in mehreren Sprachversionen an und reagieren auf diese Präferenzen; in vielen Browsern können Sie deshalb Ihre bevorzugten Sprachen einstellen.
- `User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.01)`: Dies ist die Selbstidentifikation des Browsers; in diesem Fall ist es der Internet Explorer 6.0 unter Windows XP.
- `Host: www.oreilly.de`: Bei einer HTTP/1.1-Anfrage ist der Host-Header eine Pflichtangabe: Da unter einer IP-Adresse mehrere *virtuelle Hosts* betrieben werden, muss der Server wissen, welche Website eigentlich angesprochen wurde.
- `Connection: Keep-Alive`: Der Browser bittet den Server, die Verbindung für weitere Anfragen offen zu halten. Da die meisten HTML-Dokumente zahlreiche eingebettete Bilder, SWF-Dateien und so weiter enthalten, die der Browser zusätzlich anfordern muss, beschleunigt dies den Download-Prozess.

Wenn der Server unter dem angeforderten Pfadnamen eine Datei findet, sendet er eine HTTP-Antwort an den Client zurück, die wiederum einige Header, anschließend eine Leerzeile und zum Schluss das gewünschte Dokument enthält. Diese Antwort (mit gestutztem HTML-Code des Dokuments) sieht etwa so aus:

```
HTTP/1.1 200 OK
Date: Fri, 02 Dec 2005 17:35:10 GMT
Server: Apache/1.3.28 (Linux/SuSE) mod_ssl/2.8.15 OpenSSL/0.9.7b mod_perl/1.28
Last-Modified: Fri, 02 Dec 2005 02:24:00 GMT
ETag: "7d500-3737-40e4c740"
Accept-Ranges: bytes
Content-Length: 14135
Connection: keep-alive
Content-Type: text/html

<?xml version=1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">

<head>

<!-- product id: flashmxbas2ger -->

<title>Praxiswissen Flash 8
</title>
[...]
```

Die erste Zeile gibt zunächst wieder die HTTP-Protokollversion an. Darauf folgt ein Statuscode; in diesem Fall `200 OK` für eine gefundene und korrekt gelieferte Ressource. Ein anderer bekannter Statuscode ist `404 Not Found`; Sie bekommen ihn oft bei falsch geschriebenen URLs oder fehlerhaften Links zu Gesicht.

Auch über die Server-Antwort-Header erhalten Sie hier einen kurzen Überblick:

- `Date: Fri, 02 Dec 2005 17:35:10 GMT`: Datum und Uhrzeit, als die Serverantwort abgeschickt wurde.
- `Server: Apache/1.3.28 (Linux/SuSE) mod_ssl/2.8.15 OpenSSL/0.9.7b mod_perl/1.28`: Die Selbstidentifikation der Server-Software. Über Apache erfahren Sie im nächsten Kapitel einiges.
- `Last-Modified: Fri, 02 Dec 2005 02:24:00 GMT`: Datum und Uhrzeit der letzten Änderung; wichtig für das Caching (die Zwischenspeicherung) des Dokuments in Browsern und Proxy-Servern.
- `ETag: "7d500-3737-40e4c740"`: Eine Art Identitätsstempel des Dokuments.
- `Accept-Ranges: bytes`: Der Server gibt bekannt, dass er Dateien nicht nur am Stück, sondern auch byteweise liefern kann. Dies nutzt beispielsweise der Acrobat Reader für lange PDF-Dokumente.
- `Content-Length: 14135`: Länge der gelieferten Datei in Byte.

- `Connection: keep-alive`: Der Server bestätigt die Anforderung einer bleibenden Verbindung.
- `Content-Type: text/html`: Datentyp des gelieferten Dokuments; hier handelt es sich um ein HTML-Dokument.

Selbstverständlich läuft diese Kommunikation normalerweise vollständig im Hintergrund ab. Es ist aber eine unschlagbare Stärke der meisten Internet-Kommunikationsprotokolle, dass sie wie HTTP für Menschen lesbaren Klartext verwenden, denn dadurch können Programmierer und Administratoren Schwierigkeiten leicht erkennen.

Wenn die `GET`-Anfrage Formulardaten transportieren soll, wird an die eigentliche URL ein so genannter *Query-String* angehängt: Hinter einem Fragezeichen stehen mehrere, durch `&`-Zeichen oder Semikola (`;`) getrennte `Name=Wert`-Paare. Leerzeichen in den Werten werden dabei durch `+` ersetzt; zahlreiche weitere Sonderzeichen werden durch ein Prozentzeichen und ihren hexadezimalen Zeichencode dargestellt (etwa `%FC` für `ü` oder `%21` für `!`).

Angenommen, ein Benutzer gibt in ein Feld mit der Bezeichnung `name` den Namen *Günther Schmitz* und unter `alter` den Wert *42* ein. Das Ganze wird mit Hilfe einer GET-Anfrage an */bestell/send.php* verschickt. Die erste Zeile dieser Anfrage sieht so aus:

```
GET /bestell/send.php?name=G%FCnther+Schmitz;alter=42 HTTP/1.1
```

Wenn Formulardaten dagegen über eine `POST`-Anfrage versandt werden, dann werden sie nicht an die URL angehängt, sondern im *Body* der Anfrage transportiert. Dies ist der Bereich, der durch eine Leerzeile von der Anfrage und ihren Headern getrennt wird. Der Versand der obigen Daten mit `POST` sieht etwa so aus:

```
POST /bestell/send.php
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.01)
Host: www.e-commerce.net
Content-length: 31
Content-type: application/x-www-form-urlencoded

name=G%FCnther+Schmitz;alter=42
```

Genau wie eine HTTP-Antwort enthält diese Anfrage nun auch die oben beschriebenen Header `Content-length` und `Content-type`. Der Standardtyp `application/x-www-form-urlencoded` entspricht dabei – wie Sie an den Daten selbst erkennen können – der URL-Codierung beim GET-Versand. Einer der Vorteile von POST ist allerdings, dass Sie auch andere Datentypen wählen können; wenn Formulare ein Feld für den Versand lokaler Dateien enthalten, wird zum Beispiel `multipart/form-data` benutzt.

`POST` besitzt noch weitere Vorteile: Zum einen können die Formulardaten beliebig umfangreich sein. Wenn Sie in Ihren Formularen mehrzeilige Textfelder für längere Kommentare einsetzen möchten, ist `GET` nicht brauchbar, weil für URLs eine Höchst-

länge von 2000 Zeichen gilt. Abgesehen davon garantiert `POST`, dass die Anfrage auf jeden Fall vom Server verarbeitet wird. `GET`-Anfragen können dagegen auf Proxy-Servern oder im Browser-Cache zwischengespeichert werden, so dass Sie möglicherweise das erwartete Ergebnis erhalten, ohne dass der Server die Daten erneut empfangen hat. Bei Transaktionen wie etwa einer Online-Bestellung wäre dies sehr ärgerlich.

Wozu ist `GET` aber gut, wenn `POST` so viele Vorteile besitzt? Der besondere Vorteil von `GET` besteht eben darin, dass der Query-String ein Teil der URL ist. Angenommen, Sie suchen bei Google nach `"flash 8"`. Nachdem Sie auf *Suche* geklickt haben, sehen Sie im Browser folgende Adresse: *http://www.google.de/search?sourceid=navclient&hl=de&ie=UTF-8&oe=UTF-8&q=%22flash+8%22*. Diese URL kann nun zum Beispiel in die Lesezeichen-/Favoritensammlung Ihres Browsers aufgenommen werden. Auf diese Weise haben Sie auf Knopfdruck stets das aktuelle Suchergebnis zur Hand.

HTML-Formulare

HTML-Formulare bilden seit Jahren das Kernstück von Webanwendungen. Suchmaschinen, E-Commerce-Anbieter, Foren-Communities oder Webmail-Dienste können ohne Formulare nicht funktionieren. Obwohl es in diesem Buch nicht um HTML geht, erhalten Sie hier eine kurze Übersicht über HTML-Formulare, weil sie für das Verständnis von Flash-Formularen hilfreich sind.

Grundsätzlich stehen HTML-Formulare zwischen den Tags `<form>` und `</form>`. Das `<form>`-Tag besitzt eine Reihe wichtiger Attribute:

- `action="URL"`: Gibt an, an welche URL die Formulardaten versandt werden sollen. In aller Regel handelt es sich um ein serverseitiges Skript, das die Formulardaten entgegennimmt, weiterverarbeitet und anschließend eine Antwortseite generiert, die der Browser als Nächstes anzeigt. Solche Skripte lernen Sie im nächsten Kapitel am Beispiel PHP kennen.



In älterer Literatur zu HTML wird oft die Angabe einer `"mailto:"`-URL als `action`-Wert empfohlen, also etwas in dieser Form:

```
<form action="mailto:ich@meinprovider.de">
```

Dies soll den Formularinhalt per E-Mail versenden, wobei das lokale Mail-Programm des Benutzers aufgerufen wird. Aber ausgerechnet in einer der häufigsten Kombinationen aus Browser und Mailprogramm (Internet Explorer und Outlook Express) wird dies als Hyperlink auf eine E-Mail-Adresse missverstanden: Bei Klick auf den Absenden-Button öffnet sich ein Mail-Fenster ohne Inhalt. Also gilt: Selbst für das Weiterleiten der Formulardaten per E-Mail wird in der Praxis eine serverseitige Anwendung benötigt.

- `method="post"|"get"`: Mit Hilfe des Attributs `method` wird eine der beiden weiter oben ausführlich beschriebenen Versandmethoden ausgewählt.

- `enctype="MIME-Type"`: Bei Verwendung der Methode `POST` können Sie hier den Datentyp angeben. Standard ist `application/x-www-form-urlencoded` (URL-Codierung); wenn Sie `file`-Felder zum Dateiversand verwenden möchten, müssen Sie dagegen `multipart/form-data` angeben.

Zwischen `<form>` und `</form>` können Sie verschiedene Formular-Bedienelemente wie Textfelder oder Schaltflächen einfügen; daneben sind aber auch »normale« HTML-Elemente zulässig, beispielsweise Tabellen zur Formatierung oder normaler Text für die Beschriftung.

Die meisten Eingabeelemente werden durch das Tag `<input>` bereitgestellt, in dem das Attribut `type` bestimmt, um welche Sorte von Eingabeelement es sich handelt. Damit dieser Abschnitt nicht ausufert, lernen Sie hier lediglich fünf Arten von `<input>`-Objekten kennen. Beachten Sie aber, dass es noch weitere Formularelemente gibt, sowohl andere `<input>`-Typen als auch Elemente, die mit Hilfe anderer Tags erzeugt werden.

Der wichtigste Bestandteil eines Formulars ist der Absendeknopf. Wenn er angeklickt wird, werden die Formulardaten an die im `<form>`-Tag angegebene `action`-URL versandt. Das zugehörige `<input>`-Tag sieht folgendermaßen aus:

```
<input type="submit" value="Beschriftung" />
```

Das folgende Beispiel trägt die Beschriftung *Abschicken*:

```
<input type="submit" value="Abschicken" />
```

Viele Formulare enthalten neben dem Absendeknopf auch noch eine Schaltfläche zum Zurücksetzen der Eingaben. Dieser wird folgendermaßen erzeugt:

```
<input type="reset" value="Beschriftung" />
```

Hier ein Beispiel mit der Beschriftung *Zurücksetzen*:

```
<input type="reset" value="Zur&uuml;cksetzen" />
```

Am häufigsten werden sicherlich einfache Texteingabefelder benutzt. Ihre Syntax sieht so aus:

```
<input type="text" name="Feldname" />
```

Der *`Feldname`* sollte im gesamten Formular einmalig sein. Beim Absenden des Formulars wird der Inhalt des Textfeldes als *`Feldname=Eingabetext`* versandt. Neben `name` gibt es für Textfelder noch einige interessante Zusatzattribute: Mit `value="`*`Vorgabetext`*`"` können Sie einen Text angeben, der bereits beim Laden der Seite im Feld stehen soll. `size=`*`n`* bestimmt, dass das Feld *`n`* Zeichen breit sein soll, während `maxlength=`*`n`* festlegt, dass maximal *`n`* Zeichen eingegeben werden können. Das folgende Beispiel enthält den Vorgabetext *toll*, die Breite 30 Zeichen und definiert eine maximale Eingabemenge von 20 Zeichen:

```
Wie finden Sie diese Seite?
<input type="text" name="meinung" value="toll" size="30" maxlength="30" />
```

Wenn Sie statt `type="text"` das Attribut `type="password"` angeben, erhalten Sie übrigens ein Kennwortfeld. Es besitzt dieselben Attribute und Möglichkeiten wie ein gewöhnliches Textfeld, aber statt der eigentlichen Zeichen werden *** angezeigt, Beispiel:

```
Ihr Passwort bitte:
<input type="password" name="pass" size="20" maxlength="8" />
```

Schließlich sind zwei verschiedene Sorten von Ankreuzfeldern interessant: Bei *Optionsfeldern* kann aus einer Gruppe jeweils nur einer ausgewählt werden; die englische Bezeichnung *Radio Button* bezieht sich auf das entsprechende Verhalten bei den Wellenbereichsknöpfen alter Radios. *Auswahlfelder* (engl. *checkboxes*) ermöglichen dagegen die Auswahl mehrerer Optionen.

Bei diesen Elementen wird das Attribut `name` für die gesamte Gruppe mit demselben Namen belegt, während `value` den Wert angibt, der versandt wird, wenn der betreffende Button aktiviert ist. Radio Buttons besitzen also folgendes Syntaxschema:

```
<input type="radio" name="gruppenname" value="auswahl1" />
<input type="radio" name="gruppenname" value="auswahl2" />
```

Das folgende Beispiel ermöglicht die Auswahl einer Zahlungsart in einem Bestellformular:

```
Wie m&ouml;chten Sie zahlen?
<input type="radio" name="payment" value="uebw" /> &Uuml;berweisung
<input type="radio" name="payment" value="schk" /> Verrechnungsscheck
<input type="radio" name="payment" value="last" /> Lastschrifteinzug
<input type="radio" name="payment" value="kred" /> Kreditkarte
```

Wenn beispielsweise das Feld neben *Überweisung* angekreuzt wird, dann versendet der Browser mit den Formulardaten den Wert `payment=uebw`.

Die Syntax einer Checkbox-Gruppe sieht folgendermaßen aus:

```
<input type="checkbox" name="gruppenname" value="feature1" />
<input type="checkbox" name="gruppenname" value="feature2" />
```

Im folgenden Beispiel kann ein Benutzer wählen, für welche Angebote er sich interessiert – hier ist der Anbieter natürlich auch an Mehrfachauswahlen interessiert:

```
Wof&uuml;r interessieren Sie sich besonders?
<input type="checkbox" name="int" value="agit" /> Akustik-Gitarren
<input type="checkbox" name="int" value="egit" /> E-Gitarren
<input type="checkbox" name="int" value="bass" /> B&auml;sse
<input type="checkbox" name="int" value="zubh" /> Zubeh&ouml;r
```

Wenn jemand *Bässe* und *Zubehör* auswählt, wird Folgendes übermittelt: `int=bass;int=zubh`.

In Beispiel 9-1 sehen Sie den HTML-Code eines Dokuments, das ein Formular mit allen hier vorgestellten Elementen enthält. Damit Sie auch das Versenden ausprobieren können, ohne ein Serverskript zu benutzen, ist die Action-URL `#` – dieser

Wert steht für die aktuelle Seite selbst. Wenn Sie das Formular abschicken, sehen Sie die eingegebenen Werte als URL-Anhang im Adressfeld Ihres Browsers.

Beispiel 9-1: Ein vollständiges HTML-Formularbeispiel

```
<?xml version=1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
  <head>
    <title>FlashRock Music Shop - Kundenumfrage</title>
  </head>
  <body>
    <h1>FlashRock Music Shop</h1>
    <h2>Kundenumfrage 2006</h2>
    Bitte beantworten Sie kurz die folgenden Fragen,
    damit wir unseren Service f&uuml;r Sie noch weiter
    verbessern k&ouml;nnen!<br />
    <br />
    Unter allen Besuchern, die bis zum 30.06.2006
    teilnehmen, verlosen wir drei Original <b>Fender
    Stratocaster</b>-Gitarren!<br />
    <br />
    <form action="#" method="get">
      <b>Welcher Bereich unserer Website gef&auml;llt Ihnen
      am besten?</b><br />
      <input type="radio" name="bgut" value="news" /> Neuigkeiten<br />
      <input type="radio" name="bgut" value="shop" /> Shop<br />
      <input type="radio" name="bgut" value="frum" /> Forum<br />
      <input type="radio" name="bgut" value="kald" />
          Veranstaltungskalender<br />
      <br />
      <b>An welchen Bereichen unserer Site k&ouml;nnten wir
      noch etwas verbessern?</b><br />
      <input type="checkbox" name="bschl" value="news" />
          Neuigkeiten<br />
      <input type="checkbox" name="bschl" value="shop" />
          Shop<br />
      <input type="checkbox" name="bschl" value="frum" />
          Forum<br />
      <input type="checkbox" name="bschl" value="kald" />
          Veranstaltungskalender<br />
      <br />
      <b>Ihr Verbesserungsvorschlag:</b><br />
      <input type="text" name="impr" size="30" /><br />
      <br />
      <b>Wie haben Sie von uns erfahren?</b><br />
      <input type="radio" name="kont" value="pwrb" />
          Zeitschriftenwerbung<br />
      <input type="radio" name="kont" value="bann" />
          Web-Werbebanner<br />
      <input type="radio" name="kont" value="such" />
          Suchmaschine<br />
```

Beispiel 9-1: Ein vollständiges HTML-Formularbeispiel (Fortsetzung)

```
      <input type="radio" name="kont" value="empf" />
          Pers&ouml;nliche Empfehlung<br />
      <br />
      <b>Ihre E-Mail-Adresse f&uuml;r das Gewinnspiel:</b><br />
      <input type="text" name="mail" /><br />
      (wird garantiert nur f&uuml;r die Verlosung verwendet)<br />
      <br />
      <input type="submit" value="Abschicken" />
      <input type="reset" value="Zur&uuml;cksetzen" />
    </form>
  </body>
</html>
```

Speichern Sie das Formular als Dokument mit der Endung `.html` ab. Öffnen Sie es anschließend in Ihrem Browser. Das Ergebnis sollte so aussehen wie in Abbildung 9-1.

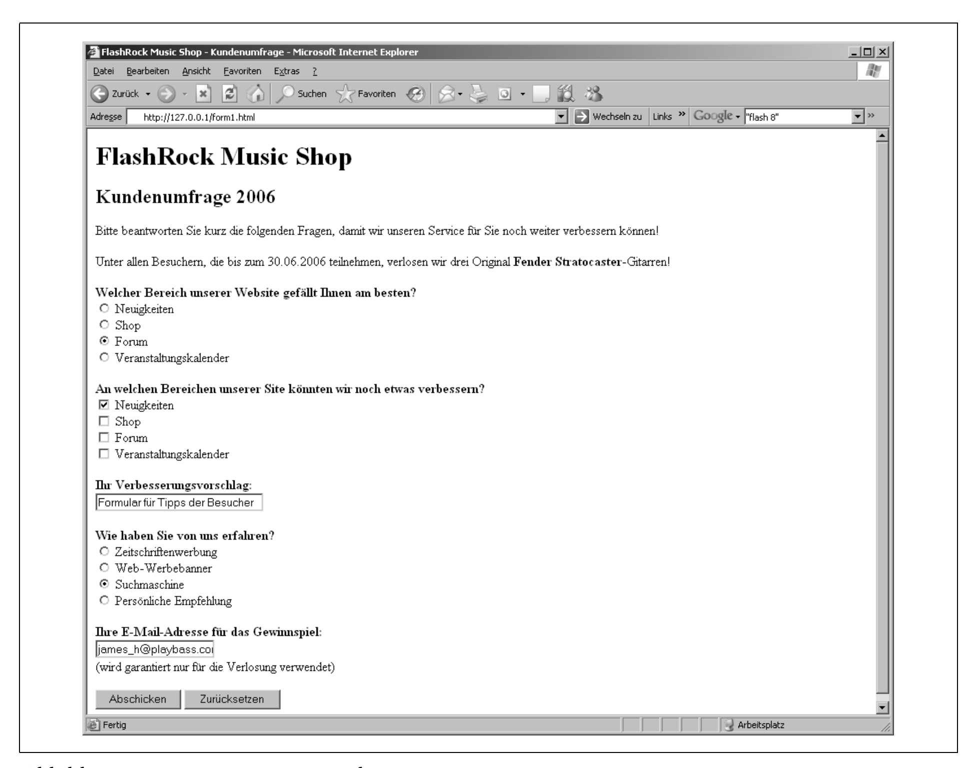


Abbildung 9-1: Das HTML-Formular im Browser

Wenn das Formular so ausgefüllt wird wie in der Abbildung, sieht die Adresszeile nach dem Klick auf die Schaltfläche *Abschicken* so aus:

```
.../form.html?bgut=frum&bschl=news&impr=Formular+f%FCr+Tipps+
der+Besucher&kont=such&mail=james_h@playbass.com
```

Der Pfad zum Dokument am Anfang der URL sieht je nach Speicherort und Browser unterschiedlich aus, so dass er hier absichtlich weggelassen und durch `...` gekennzeichnet wurde.

Die Arbeitsweise von Flash-Formularen

Flash-Formulare können genauso wie die soeben vorgestellten HTML-Formulare funktionieren: Die Funktion `getURL()`, die in anderem Zusammenhang bereits in Kapitel 8 vorgestellt wurde, dient nicht nur der »normalen« Web-Verlinkung, sondern ist auch zum Formularversand in der Lage. Zu diesem Zweck gibt es den optionalen dritten Parameter *`Methode`*:

```
getURL (URL, Fenster, Methode);
```

URL entspricht in diesem Fall dem Attribut `action` eines HTML-Formulars: Der Parameter enthält die URL, an die die Formulardaten gesendet werden sollen. *`Fenster`* müssen Sie in diesem Fall leider auch dann angeben, wenn das Ziel des Links kein anderes Fenster oder Frame sein soll, weil Flash die Parameter nach der Reihenfolge unterscheidet. Geben Sie einfach `"_self"` an, um das aktuelle Fenster beziehungsweise Frame anzusprechen. Der eigentlich wichtige Parameter für den Formularversand ist natürlich *`Methode`*; die beiden möglichen Werte sind `"GET"` oder `"POST"`. Verwenden Sie für besonders umfangreiche Datenmengen gemäß der obigen Beschreibung `"POST"`, ansonsten können Sie auch `"GET"` benutzen.

Das Praktische ist, dass Sie gar nicht einmal unbedingt besondere »Formularfelder« zu definieren brauchen – wenn eine solche `getURL()`-Anweisung ausgeführt wird, versendet der Flash Player einfach sämtliche Variablen, die im aktuellen Kontext bekannt sind, als `Variable=Wert`-Paare. Deshalb können Sie für die Formulareingabe und -auswahl nicht nur vorgefertigte Komponenten wie in HTML-Formularen verwenden, sondern auch beliebige Schaltflächen und Movieclips erstellen, die die Werte von Variablen ändern. Auch für die Texteingabe ist gesorgt: Flash stellt Textfelder zur Verfügung, die mit einer Variablen verknüpft sind – was ein Benutzer dort eingibt, ist automatisch der Wert dieser Variablen.

Bevor es im nächsten Abschnitt ins Detail geht, sollten Sie das folgende kurze Beispiel durcharbeiten: Die Eingabe in ein Textfeld wird auf Knopfdruck als Suchbegriff an Google weitergereicht. Sehen Sie sich zu diesem Zweck einmal an, wie die URL einer ausgeführten Google-Suche aussieht. Beispielsweise besitzt die Suche nach *Flash* folgende URL:

```
http://www.google.de/search?q=flash&ie=UTF-8&hl=de&meta=
```

Das Haupt-Suchskript *search* erwartet also offensichtlich einen Query-String mit dem Feld `q=`*`Suchbegriff`* (durch Tests mit direkter Eingabe lässt sich herausfinden, dass die restlichen Felder nicht notwendig sind). In Flash lässt sich diese Anforderung durch ein Texteingabefeld realisieren, dem der Variablenname `q` zugeordnet

wird. Anschließend wird eine Schaltfläche erstellt, die die Beschriftung *Google-Suche* sowie ein Skript mit der passenden `getURL()`-Anweisung erhält. Gehen Sie im Einzelnen wie folgt vor:

1. Wählen Sie das *Textwerkzeug* aus der Werkzeugpalette. Stellen Sie links oben in der Eigenschaftenleiste den Texttyp *Eingabetext* ein. Ziehen Sie nun auf der Bühne ein Textfeld mit der Schriftarteinstellung *_sans* (Geräteschriftart) auf. Geben Sie in der Eigenschaftenleiste unter *Var:* den Variablennamen `q` ein.
2. Erstellen Sie ein Schaltflächensymbol mit der Beschriftung *Google-Suche*. Ziehen Sie eine Instanz davon auf die Bühne, und weisen Sie ihr folgendes Skript zu:

```
on (release) {
   getURL ("http://www.google.de/search", "_self", "GET");
}
```

3. Erstellen Sie mit Hilfe der Funktion *Veröffentlichen* (oder auf Wunsch manuell) ein HTML-Dokument mit der eingebetteten SWF-Version dieses Films. Öffnen Sie die HTML-Datei im Browser, geben Sie einen Suchbegriff in das Textfeld ein, und drücken Sie auf die Schaltfläche. Im Browser erscheint nun das Ergebnis der Google-Suche nach dem eingegebenen Begriff.

Technisch gesehen handelt es sich hier um das Äquivalent eines HTML-Formulars mit der Versandmethode `GET`. Beim Klick auf die Schaltfläche konstruiert der Flash Player aus dem Namen der Variablen (`q`) und ihrem Wert (der Benutzereingabe) einen Query-String, hängt ihn an die URL der `getURL()`-Funktion an und sendet die Anfrage ab. Bei der Suchmaschine kommt dies genau so an, als hätten Sie einen Suchbegriff in deren eigenes Textfeld eingetippt.

Übrigens versenden auch die Funktionen `loadVariablesNum()` beziehungsweise `loadVariables()` optional Variablen, wenn Sie als drittes Argument eine der Methoden `"GET"` oder `"POST"` hinzufügen. Auf diese Weise können Sie die Variablen an eine Webanwendung versenden und *innerhalb* des aktuellen Films eine Antwort empfangen – deshalb ist es in der Praxis sogar die gängigere Methode zum Erstellen von Flash-Formularen. Im nächsten Kapitel wird dieses Thema vertieft.

Textfelder und Komponenten in Flash

Das Kernstück von Flash-Formularen bilden, wie bereits erwähnt, Texteingabefelder sowie die seit Flash MX verfügbaren Komponenten. Aus diesem Grund sollen diese Bedienelemente hier näher beleuchtet werden.

Dynamische Textfelder und Texteingabefelder

Flash 8 bietet zwei verschiedene Arten von Textfeldern an: *Dynamische Textfelder* können nicht vom Benutzer editiert werden, aber Sie können ihren Inhalt per ActionScript bestimmen. Für Formulare interessanter sind die *Texteingabefelder*, in

die Benutzer Text eintippen können. Beide Textfeldsorten werden mit Hilfe des Textwerkzeugs in der Werkzeugpalette erstellt, das bereits in Kapitel 2, *Zeichnen mit Flash*, vorgestellt wurde.

Sobald Sie das Textwerkzeug auswählen, finden Sie in der Eigenschaftenleiste zahlreiche Einstellungen für Text vor. Auch diese Optionen wurden in Kapitel 2 beschrieben. Im Pop-up-Menü links oben können Sie sich zwischen den drei Varianten *Statischer Text* (für unveränderliche Beschriftungen), *Dynamischer Text* (per ActionScript änderbare Felder) und *Eingabetext* (Texteingabefelder) entscheiden. Dies sollten Sie jeweils tun, bevor Sie eine Textbox aufziehen; die nachträgliche Änderung ist immer ein wenig lästig.

In Abbildung 9-2 sehen Sie die Eigenschaftenleiste mit den möglichen Optionen für dynamischen Text. Viele von ihnen entsprechen den bereits besprochenen Einstellungen für statischen Text; hier werden lediglich die Unterschiede erwähnt.

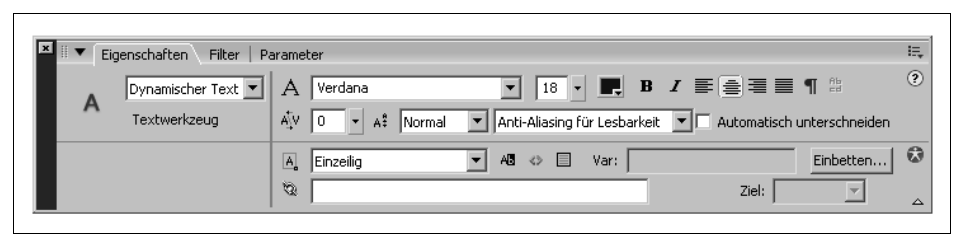


Abbildung 9-2: Einstellungen für dynamischen Text in der Eigenschaftenleiste

Links oben unter dem Texttyp können Sie dem Textfeld einen Instanznamen zuweisen. Damit können viele der in Kapitel 6 besprochenen Instanzmethoden und -eigenschaften auch bei Textfeldern benutzt werden. Daneben gibt es spezielle ActionScript-Konstrukte, die nur für Textfelder verfügbar sind; einige von ihnen werden weiter unten in diesem Abschnitt behandelt.

Die wichtigsten Sonderfunktionen finden Sie im unteren Teil der Eigenschaftenleiste, neben den Einstellungen für Position und Größe. Zunächst einmal können Sie sich per Pull-down-Menü entscheiden, ob das Feld *Einzeilig* oder *Mehrzeilig* sein soll; die spezielle dritte Option *Mehrzeilig, kein Umbruch* ist geeignet, wenn Sie Zeilenumbrüche selbst setzen möchten.

Die daneben liegende Schaltfläche *Auswählbar* bestimmt, ob Benutzer den Text auswählen und in die Zwischenablage kopieren können. Wenn Sie die nächste Schaltfläche, *Text als HTML wiedergeben*, aktivieren, können Sie der Textfeldvariablen einfache HTML-Codes zur Textformatierung zuweisen. Die Schaltfläche *Rahmen um Text zeigen* sorgt dafür, dass das Textfeld mit sichtbarem Rand und weißem Hintergrund gezeichnet wird.

Var ist die wichtigste Einstellung für ein dynamisches Textfeld: Hier wird der Name der Variablen eingetragen, mit der das Feld verbunden ist. Sobald Sie dieser Variablen per ActionScript einen Wert zuweisen, erscheint dieser automatisch im Feld.

Die Schaltfläche *Einbetten* öffnet den Dialog, den Sie in Abbildung 9-3 sehen. Wenn Sie für das Textfeld eine individuelle Schriftart benutzen möchten, dann müssen Sie hier angeben, welche Zeichen dieser Schrift mit dem SWF-Film exportiert werden sollen. Schließlich kann Flash beim Export noch nicht wissen, welche Zeichen hier später einmal benötigt werden.

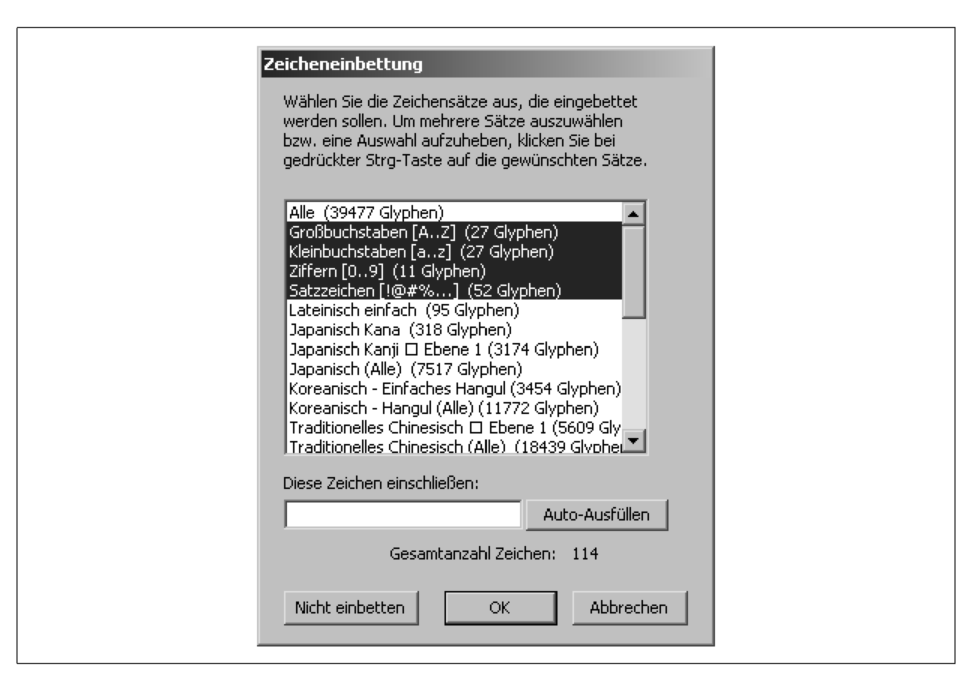


Abbildung 9-3: Auswahl der Zeichen einer Schriftart, die für ein dynamisches Textfeld exportiert werden sollen

Damit der vollständige Zeichenvorrat zur Verfügung steht, der für deutschen Text erforderlich ist, sollten Sie die folgenden Einträge anklicken: *Großbuchstaben [A..Z]*, *Kleinbuchstaben [a..z]*, *Ziffern [0..9]*, *Satzzeichen*, *Lateinisch einfach* sowie *Lateinisch I*. Verwenden Sie die **STRG**-Taste (Windows) beziehungsweise ⌘-Taste (Macintosh), um mehrere Zeilen auszuwählen.

Unter *Diese Zeichen einschließen* können Sie optional auch eine eigene Liste von Zeichen angeben – falls Ihr Feld beispielsweise einen Preis enthalten soll, genügt es, *Ziffern [0..9]* anzuklicken und unten die drei zusätzlichen Zeichen Euro (€), Komma (,) und Strich (-) anzugeben. *Auto-Ausfüllen* fügt sämtliche Zeichen hinzu, die Sie in der Arbeitsumgebung in das Textfeld geschrieben haben.

Wenn Sie Speicher und damit Ladezeit sparen möchten, können Sie stattdessen *Nicht einbetten* auswählen; als Schriftart müssen Sie dann einen der drei speziellen Einträge *_serif* (allgemeine Serifenschrift)[1], *_sans* (allgemeine serifenlose Schrift) oder *_typewriter* (Schreibmaschinenschrift mit fester Laufweite) auswählen.

Sie können Text im Textfeld markieren und neben dem Kettensymbol eine URL eintragen. Dadurch wird der markierte Text zum Hyperlink auf diese Adresse. Unter *Ziel* können Sie zusätzlich den Namen eines Browserfensters oder Frames angeben, in das die entsprechende Seite geladen werden soll. Die speziellen vorgefertigten Einträge bedeuten dabei Folgendes (siehe dazu Kapitel 8): *_self* ist das aktuelle Fenster oder Frame, *_parent* steht für das übergeordnete Frame, *_top* für das gesamte Browserfenster ohne Frames. *_blank* schließlich öffnet die verlinkte Website in einem neuen, leeren Browserfenster.

Für *Eingabetext* sind fast dieselben Einstellungen verfügbar. Hier lässt sich lediglich kein Hyperlink einstellen. Dafür können Sie die *Maximale Zeichenanzahl* begrenzen, die ein Benutzer eingeben kann. Die Voreinstellung 0 bedeutet allerdings nicht etwa *keine* Zeichen, sondern beliebig viele.

Eine Uhr im dynamischen Textfeld

Damit Sie den Umgang mit Textfeldern auch praktisch nachvollziehen können, folgt hier zunächst einmal ein kurzes Praxisbeispiel außer der Reihe: In einem dynamischen Textfeld mit HTML-Formatierung sollen Datum und Uhrzeit angezeigt werden. Dazu wird die in Kapitel 7 entwickelte Klasse `CustomDate` verwendet; stellen Sie sicher, dass sich die Datei *CustomDate.as* im aktuellen Verzeichnis befindet.

Erstellen Sie zunächst ein dynamisches Textfeld. Stellen Sie die Schriftart Arial, 20 Punkt, Schwarz ein. Aktivieren Sie die Optionen *Mehrzeilig* und *Text als HTML wiedergeben*. Wählen Sie unter *Zeichen* die *Großbuchstaben*, *Kleinbuchstaben* und *Ziffern*; geben Sie bei der Option *Diese Zeichen einschließen* zusätzlich die Zeichen Punkt (.), Komma (,) und Doppelpunkt (:) an. Geben Sie schließlich unter *Var* den Variablennamen `zeit` ein.

Erstellen Sie ein leeres Movieclip-Symbol mit dem Namen *steuerclip*, und ziehen Sie es auf die Bühne. Weisen Sie ihm das folgende Skript zu:

```
onClipEvent (enterFrame) {
   // CustomDate-Objekt erstellen
   var jetzt = new CustomDate();
   // Sonntag rot darstellen, alle anderen Tage normal
   var wt = jetzt.getInternationalDayName ("de");
```

1 Serifen sind die kleinen Häkchen an den Buchstaben mancher Schriften: Times New Roman ist eine Serifenschrift, während Arial serifenlos ist.

```
   if (wt == "Sonntag") {
      _root.zeit = "<font color=\"#FF0000\">" + wt + "</font>";
   } else {
      _root.zeit = wt;
   _root.zeit += jetzt.getInternationalFormat (", %d.%m.%Y\n%H:%M:%S", "de");
}
```

Das HTML-Tag `<font>` kann die Farbe (`color`), die Schriftart (`face`) und die Größe (`size`) der Schrift angeben. In Flash besteht eine Besonderheit darin, dass `size` eine Schriftgröße in Punkt ist, während es sich im Browser um eine Stufe von 1 bis 7 handelt. Das Konstrukt `\"` ist eine so genannte *Escape-Sequenz* – durch den vorangestellten Backslash können Sie innerhalb der Anführungszeichen das Anführungszeichen selbst als Zeichen verwenden. Auf ähnliche Weise codiert die Escape-Sequenz `\n` einen Zeilenumbruch.

Eine persönliche Begrüßung

Hier noch ein weiteres, ganz kurzes Beispiel, in dem die Verwendung eines Texteingabefeldes demonstriert wird: Nach der Eingabe seines Namens und einem Klick auf *OK* soll der Benutzer persönlich begrüßt werden.

Erstellen Sie zunächst ein Textfeld mit der Eigenschaft *Eingabetext*. Stellen Sie unter *Einbetten* am besten die volle Palette der Zeichen ein, die für Deutsch notwendig sind (siehe oben). Verknüpfen Sie dieses Feld mit der Variablen `name`.

Fügen Sie darunter ein weiteres Textfeld ein, diesmal mit der Eigenschaft *dynamischer Text*. Stellen Sie auch hier alle Zeichen für Deutsch ein; weisen Sie ihm den Variablennamen `gruss` zu.

Erstellen Sie schließlich über dem Eingabefeld die Beschriftung (statischer Text) *Ihr Name, bitte* und darunter eine *OK*-Schaltfläche. Weisen Sie Letzterer folgendes Skript zu:

```
on (release) {
   // Grussformel je nach Tageszeit
   var jetzt = new Date();
   var stunde = jetzt.getHours();
   if (stunde < 12) {
      gruss = "Guten Morgen";
   } else if ( stunde < 18) {
      gruss = "Guten Tag";
   } else {
      gruss = "Guten Abend";
   }
   // Name hinzufügen
   gruss += ", " + name + "!";
}
```

HTML in dynamischen Textfeldern

Wenn Sie die Option *HTML* aktiviert haben, können Sie einige HTML-Tags zur Formatierung dynamischer Textfelder benutzen, indem Sie der verknüpften Variablen einen Wert zuweisen, der diese enthält. Welche Tags zulässig sind und wie sie angewendet werden, sehen Sie in Tabelle 9-1. Sie können diese Tags gemäß den allgemeinen Hinweisen zu HTML in Kapitel 5 beliebig ineinander verschachteln, um ihre Wirkung zu kombinieren.

Tabelle 9-1: Zulässige HTML-Tags zur Formatierung der Inhalte dynamischer Textfelder

HTML-Tag	Erläuterung
Text	stellt den *Text* fett dar
<i>*Text*</i>	stellt den *Text* kursiv dar
<u>*Text*</u>	stellt den *Text* unterstrichen dar
Textblock	setzt den *Textblock* in einen eigenen Absatz
Text	stellt für *Text* die angegebene Schrift ein
Text	stellt für *Text* die *Schriftgröße* (in Punkt) ein
Text	stellt für *Text* die *Schriftfarbe* ein (hexadezimales RGB, z.B. *#00FF00* für Grün)

ActionScript-Eigenschaften für Textfelder

Wie oben bereits erwähnt, können Sie Textfeldern nicht nur eine Variable, sondern auch einen Instanznamen zuordnen. Dieser ist der Schlüssel zu einigen Methoden (deren Komplexität den Rahmen dieses Kapitels sprengen würde) und zahlreichen Eigenschaften. Exemplarisch sollen hier nur vier sehr nützliche Eigenschaften vorgestellt werden: Mit `tabIndex` und `tabEnabled` lässt sich die Reihenfolge beeinflussen, in der Sie mit der Tab-Taste durch Eingabefelder blättern können. `scroll` und `maxscroll` ermöglichen Ihnen dagegen das programmgesteuerte Scrollen in Feldern, die zu viel Text enthalten.

Wenn Sie ein Bild mit mehreren Texteingabefeldern erstellen, können Sie diese standardmäßig nacheinander (von links nach rechts und von oben nach unten) mit der Tabulator-Taste aktivieren. Setzen Sie die Eigenschaft `tabIndex` Ihrer Textfeld-Instanzen, um diese Reihenfolge zu modifizieren: Je niedriger der Wert von `tabIndex`, desto früher kommt ein Feld an die Reihe. Das folgende Codebeispiel geht davon aus, dass Sie Eingabefelder mit den Instanznamen `name`, `str`, `hausnr`, `plz` und `ort` erstellt haben, die in der typischen Reihenfolge einer Anschrift den Eingabefokus erhalten sollen:

```
name.tabIndex = 1;
str.tabIndex = 2;
hausnr.tabIndex = 3;
plz.tabIndex = 4;
ort.tabIndex = 5;
```



Sobald Sie anfangen, `tabIndex` einzusetzen, benötigt *jedes* Eingabefeld auf dem aktuellen Bildschirm, das per Tab erreichbar sein soll, einen individuellen Wert für diese Eigenschaft.

Wenn Sie ein bestimmtes Textfeld völlig aus der Tab-Abfolge herausnehmen möchten, können Sie dessen Eigenschaft `tabEnabled` explizit auf `false` setzen. Das folgende Beispiel sorgt dafür, dass das Feld mit dem Instanznamen `zusatz` im Hauptfilm nicht mehr durch Tab angesprungen werden kann:

```
zusatz.tabEnabled = false;
```



Neben den Texteingabefeldern können auch die im nächsten Unterabschnitt vorgestellten Komponenten in die Tab-Reihenfolge mit aufgenommen werden. Dazu besitzen auch sie die Eigenschaften `tabIndex` und `tabEnabled`.

Wenn Sie mehrzeilige Textfelder per ActionScript mit Text füllen, passt sich deren Höhe nicht an die Textmenge an, wie es in der Flash-Arbeitsumgebung der Fall ist. Stattdessen wird immer nur ein Teil des Textes angezeigt. Standardmäßig sind Flash-Textfelder nicht mit Rollbalken zum Erreichen des restlichen Textes ausgestattet. Dafür können Sie sich selbst individuelle Scroll-Bedienelemente programmieren, indem Sie die Eigenschaften `scroll` und `maxscroll` eines Textfeldes[2] benutzen: `scroll` bestimmt, welche Zeile des Gesamttextes zurzeit am oberen Rand des Feldes angezeigt wird, während `maxscroll` die Nummer der letzten möglichen Zeile angibt.

Wenn Sie einer Schaltfläche folgenden Code zuweisen, wird der Text des Textfeldes `vielText` bei jedem Klick um eine Zeile nach oben gerollt, bis die letzte Zeile erreicht ist:

```
on (release) {
   if (vielText.scroll < vielText.maxscroll) {
      vielText.scroll++;
   }
}
```

Eine Schaltfläche für die andere Richtung könnten Sie dagegen mit diesem Code ausstatten:

```
on (release) {
   if (vielText.scroll > 1) {
      vielText.scroll--;
   }
}
```

2 Neben `scroll` und `maxscroll` besitzen Textfelder noch einige weitere Optionen, die sich auf das Scrollen beziehen; dies würde hier allerdings zu weit führen.

Noch praktischer sind natürlich Schaltflächen, die weiterscrollen, solange sie gedrückt bleiben und solange weiterer Text vorhanden ist. Erstellen Sie zu diesem Zweck einen Steuerclip, der folgenden Code enthält (unter der Voraussetzung, dass sich `vielText` im Hauptfilm befindet):

```
onClipEvent (enterFrame) {
    if (_root.scrollen == -1) {
        if (_root.vielText.scroll > 1) {
            _root.vielText.scroll--;
        }
    } else if (_root.scrollen == 1) {
        if (_root.vielText.scroll < _root.vielText.maxscroll) {
            _root.vielText.scroll++;
        }
    }
}
```

Der Button für das Rollen nach oben erhält nun folgende neue Aktion, die nur noch die Variable `scrollen` beeinflusst:

```
on (press) {
    scrollen = -1;
}

on (release) {
    scrollen = 0;
}
```

Im Skript des anderen Buttons wird die Variable beim Drücken entsprechend auf 1 und beim Loslassen wieder auf 0 gesetzt:

```
on (press) {
    scrollen = 1;
}

on (release) {
    scrollen = 0;
}
```

Komponenten

Seit der Version Flash MX stehen die so genannten *UI-Komponenten* (UI bedeutet User Interface, also Benutzeroberfläche) zur Verfügung. Anders als die üblichen Flash-Elemente entsprechen sie der Optik und Funktionsweise von Bedienelementen des Betriebssystems. Durch die Einführung der Komponenten hofft Macromedia, Flash stärker als Entwicklungsplattform für professionelle Webanwendungen (*Rich Internet Applications*) zu etablieren.

Sie finden die Komponenten im gleichnamigen Bedienfeld (*Fenster → Komponenten* oder **STRG + F7**). In Abbildung 9-4 wird diese Palette dargestellt.

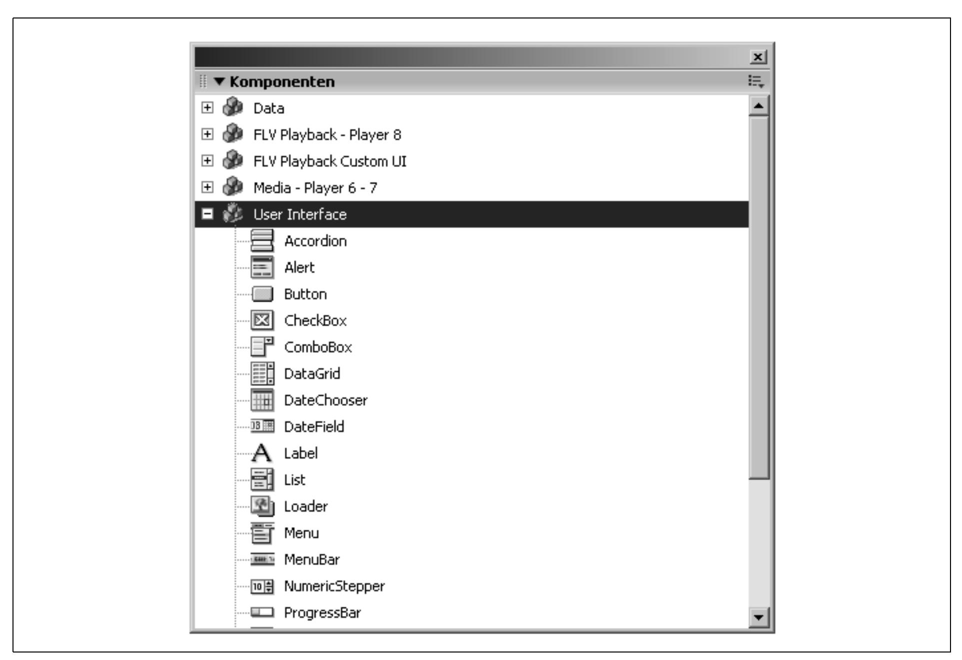


Abbildung 9-4: Das Bedienfeld »Komponenten«, hier aus Flash Professional 8

Nachdem Sie eine Komponente auf die Bühne gezogen haben, müssen Sie ihre Einstellungen anpassen. Neben dem bekannten Instanznamen, der wie immer in der Eigenschaftenleiste eingegeben wird, gibt es dazu den *Komponenten-Inspektor* (*Fenster → Komponenten-Inspektor* oder **ALT + F7**, siehe Abbildung 9-5). Die Registerkarte *Parameter* in der Eigenschaftenleiste erfüllt übrigens dieselbe Aufgabe.

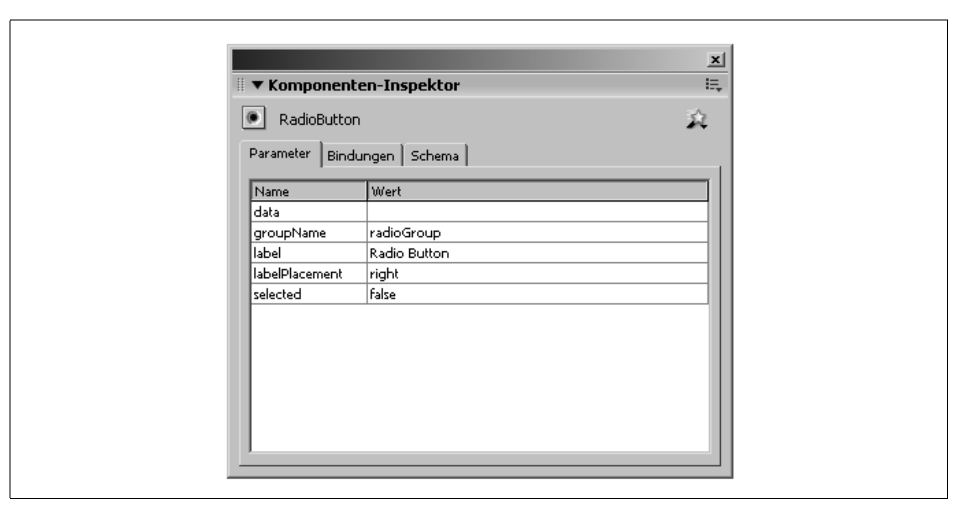


Abbildung 9-5: Der Komponenten-Inspektor, hier mit den Eigenschaften eines RadioButtons

Die UI-Komponenten, die Flash 8 ab Werk enthält, sind in Tabelle 9-2 zu finden. Flash Professional 8 ist noch mit zahlreichen weiteren Komponenten ausgestattet, die nicht nur die grafische Oberfläche betreffen, sondern beispielsweise auch die automatische Anbindung an externe Datenquellen. Zudem gibt es Komponenten von Drittanbietern, mit denen Sie den Funktionsumfang von Flash nochmals erweitern können.

Tabelle 9-2: Die vorgefertigten UI-Komponenten von Flash 8

Komponente	Bedeutung	ActionScript-Grundlagen
Button	einfache Schaltfläche	`on (click) {...}` verarbeitet Klick-Ereignis
Checkbox	Auswahlfeld	*`Instanzname`*`.selected` ist `true`, wenn angekreuzt, sonst `false`
ComboBox	Pull-down-Menü mit Texteingabe	*`Instanzname`*`.text` enthält den ausgewählten bzw. eingegebenen Text
Label	zusätzliche Beschriftung	–
List	Liste zur Auswahl eines Eintrags	*`Instanzname`*`.selectedIndex` liefert gewählte Zeilennummer
Loader	Anzeigefeld für SWF- oder JPEG-Dateien	*`Instanzname`*`.contentPath` kann die URL des Inhalts zugewiesen werden
NumericStepper	Auswahlfeld für das Durchblättern von Zahlenfolgen	*`Instanzname`*`.value` gibt den aktuell gewählten Wert zurück
ProgressBar	Fortschrittsbalken	*`Instanzname`*`.source` gibt das Objekt an, dessen Ladevorgang angezeigt wird
RadioButton	Optionsfeld	Eigenschaft `groupName`: gemeinsamer Gruppenname für mehrere RadioButtons. *`Gruppenname`* `.selectedData` gibt den Datenwert des ausgewählten Buttons aus der Gruppe zurück
ScrollPane	Anzeigefeld mit automatischen Rollbalken	siehe `Loader`
TextArea	mehrzeiliges Textfeld	*`Instanzname`*`.text` ist der Textinhalt, der zugewiesen oder ausgelesen werden kann
TextInput	einzeiliges Textfeld	siehe `TextArea`
Window	frei schwebendes Fenster mit Titelleiste und Schließfeld	siehe `Loader`

Hier nur ein kleines Beispiel für die Anwendung von Komponenten: Im Ausgabefenster wird die Auswahl aus einer `RadioButton`-Gruppe sowie der Text aus einer `ComboBox` ausgegeben, sobald eine `Button`-Komponente angeklickt wird.

Ziehen Sie zunächst drei `RadioButton`-Komponenten auf die Bühne. Klicken Sie alle drei an, und weisen Sie ihnen im Komponenten-Inspektor folgende Eigenschaften zu: `groupName` (Gruppenname) bei allen *interesse*, unter `data` (Datenwert bei Auswahl) *egitarre*, *agitarre* beziehungsweise *bass*. Die Eigenschaft `label` (Beschriftung) wird schließlich entsprechend mit den drei Werten *E-Gitarren*, *Akustische Gitarren*

und *Bässe* versehen. Falls eine Beschriftung nicht passt, müssen Sie die Breite der entsprechenden Komponente per Eigenschaftenleiste oder Info-Palette ändern.

Fügen Sie über den RadioButtons die folgende Frage als statischen Text oder `Label`-Komponente ein: *Welche Instrumente interessieren Sie am meisten?* Erstellen Sie unter der RadioButton-Gruppe die nächste statische Beschriftung: *Wie haben Sie von unserer Site erfahren (auswählen oder eintragen)?*

Ziehen Sie nun eine `ComboBox`-Komponente auf die Bühne. Weisen Sie ihr in der Eigenschaftenleiste den Instanznamen `siteInfo` zu. Stellen Sie im Komponenten-Inspektor die Eigenschaft `editable` auf *true* – das bedeutet, dass neben der Auswahl der vorgefertigten Einträge auch neuer Text eingetippt werden kann. Klicken Sie anschließend rechts im Feld der Eigenschaft `labels` auf das kleine Lupensymbol. Es erscheint der Dialog *Werte*; geben Sie hier mit Hilfe der Schaltfläche + folgende Texte ein: *Zeitschriftenwerbung*, *Online-Werbung*, *Andere Website* und *Persönliche Empfehlung*.

Schließlich wird noch eine Komponente vom Typ `Button` benötigt. Weisen Sie ihr über den Komponenten-Inspektor das `Label` *Abschicken* zu. Anschließend erhält die Instanz folgendes Skript:

```
on (click) {
    // RadioButton-Auswahl ausgeben
    trace ("Ihr Interesse: " + _root.interesse.selectedData);
    // ComboBox-Text ausgeben
    trace ("Ihr Kontakt: " + _root.siteInfo.text);
}
```



Wenn Sie vorhaben, die Benutzerauswahl aus einer Komponente mit einem Formular an ein serverseitiges Skript zu versenden, dann müssen Sie diese Auswahl in einer normalen Variablen speichern. Beispielsweise könnten Sie die RadioButton-Auswahl aus dem obigen Beispiel folgendermaßen behandeln, um sie als Formularvariable namens `instr` zu verschicken:

```
var instr = _root.interesse.selectedData;
```

Praxisbeispiel: Ein Bestellformular

Damit die Site des FlashRock Music Shop tatsächlich zum echten Online-Shop wird, muss sie natürlich ein Bestellformular enthalten. Deshalb ist das Praxisbeispiel dieses Kapitels die Erstellung der Benutzeroberfläche des Formulars. Die Bestellannahme und -verarbeitung, die auf dem Server stattfindet, wird übrigens im nächsten Kapitel vorgestellt.

Öffnen Sie die Datei *form_start.fla* im Verzeichnis *beispiele/kapitel9* der CD-ROM. Ihre Aufgabe besteht darin, mehrere Texteingabefelder, Schaltflächen und ActionScript-Anweisungen hinzuzufügen; der statische Teil der Benutzeroberfläche ist dagegen bereits fertig.

Setzen Sie den Abspielkopf zunächst in das Bild mit der Bezeichnung *artikel* (Bild 2). Dieses Frame bildet die erste Formularseite, auf der die eigentliche Bestellauswahl stattfindet: Jeder Artikel soll ein Eingabefeld für die gewünschte Stückzahl erhalten. Wählen Sie dazu in der Werkzeugpalette das Textwerkzeug aus. Stellen Sie in der Eigenschaftenleiste den Typ *Eingabetext*, die Schriftart Arial, die Größe 14 Punkt und die Farbe Schwarz ein. Aktivieren Sie die Schaltfläche *Rahmen um Text zeigen*. Wählen Sie unter *Zeichen* ausschließlich Ziffern aus – etwas anderes soll in diese Felder gar nicht eingegeben werden.

Ziehen Sie das erste Textfeld unter der Beschriftung Stückzahl in der Zeile *Original Fender Stratocaster E-Gitarre* auf. Geben Sie in der Eigenschaftenleiste unter *Var.* den Variablennamen `strat` ein. Erstellen Sie darunter (beispielsweise durch Duplizieren) die vier restlichen Textfelder mit folgenden Variablennamen: `yama` (*Yamaha Halbakustik-Gitarre*), `amp` (*Marshall Kofferverstärker*), `kab` (*Gitarrenkabel*) und `buch` (*Lehrbuch »Rock-Gitarre«*).

Klicken Sie als Nächstes das Schlüsselbild auf der Ebene *actions* im aktuellen Frame an. Geben Sie hier in der Aktionen-Palette die folgende Bildaktion ein:

```
stop();
```

Aktivieren Sie nun auf der Bühne die Schaltfläche *Weiter*. Sie erhält das folgende Skript:

```
on (release) {
   gotoAndStop ("daten");
}
```

Wenn Sie alle Arbeiten im Frame *artikel* abgeschlossen haben, sollte es so aussehen wie in Abbildung 9-6.

Als Nächstes müssen die Textfelder und Skripte im Frame *daten* erstellt werden. Hier werden die persönlichen Daten zur Bestellung eingegeben; daneben wird die Zwischensumme der Bestellung angezeigt.

Erstellen Sie als Erstes neben der Beschriftung *Zwischensumme* ein Textfeld vom Typ *Dynamischer Text*. Weisen Sie diesem den Variablennamen `zsumme` zu. Stellen Sie die Schriftart *Arial*, die Größe *16 Punkt* und die Farbe Gelb (*#FFFF00*) ein; dieses Feld soll keinen sichtbaren Rahmen haben. Klicken Sie die Schaltfläche *Zeichen* an. Wählen Sie darin den Eintrag *Ziffern [0..9]*, und geben Sie unter *Diese Zeichen einschließen* zusätzlich die drei Zeichen »`-`«, »`,`« und »`.`« ein.

Fügen Sie nun einige Felder vom Typ *Eingabetext* ein (Arial 14 Punkt, schwarz, mit Rahmen): Variablenname `kunde` (für das Feld neben der Beschriftung *Name*), `adr` (für *Straße*), `plz` (*PLZ*), `ort` (*Ort*) und `nr` (*Kontonummer*). Unter *Zeichen* sollten Sie für alle diese Felder (mit gedrückter **STRG**- beziehungsweise -Taste) folgende Bereiche auswählen: *Großbuchstaben [A..Z]*, *Kleinbuchstaben [a..z]*, *Ziffern [0..9]*, *Satzzeichen*, *Lateinisch einfach* sowie *Lateinisch I*.



Abbildung 9-6: Die erste Seite des Flash-Formulars

Ziehen Sie links neben die Bezeichnung Rechnung und links neben Bankeinzug je eine Instanz des Symbols *myradio*. Es handelt sich dabei um einen Movieclip, der die Aufgabe von RadioButtons übernehmen soll, aber optisch besser zu diesem Formular passt als die offizielle Komponente. Weisen Sie den beiden Buttons die Instanznamen *r1* beziehungsweise *r2* zu. Ziehen Sie danach je eine Instanz von *leerbutton* darauf.

Dieses Frame erhält auf der Ebene *actions* das folgende Skript:

```
// Zwischensumme berechnen
zsumme = 0;
if (strat) {
  zsumme += strat * 599;
}
if (yama) {
  zsumme += yama * 399;
}
if (amp) {
  zsumme += amp * 299;
}
if (kab) {
  zsumme += kab * 14;
}
if (buch) {
  zsumme += buch * 19;
}
zsumme += ",-- .";
```

```
// Initialisierung der Zahlungsweise: Rechnung ist vorausgewählt
var pay = "rech";
r1.gotoAndStop(2);

stop();
```

Weisen Sie dem linken *leerbutton*, der auf der Instanz *r1* liegt, folgende Action-Script-Anweisungen zu:

```
on (release) {
  r1.gotoAndStop(2);
  r2.gotoAndStop(1);
  pay = "rech";
}
```

Die Schaltfläche auf *r2* erhält natürlich ein sehr ähnliches Skript:

```
on (release) {
  r1.gotoAndStop(1);
  r2.gotoAndStop(2);
  pay = "bank";
}
```

Weisen Sie nun dem Button *Zurück* die folgenden Anweisungen zu:

```
on (release) {
  gotoAndStop ("artikel");
}
```

Der Button *Bestellen*, der dem Absenden der Bestellung dient, erhält das folgende Skript:

```
on (release) {
  getURL ("order.php", "_blank", "POST");
  gotoAndStop ("danke");
}
```

Das Skript *order.php*, das die Bestellinformationen annimmt und in einem separaten Browserfenster (Ziel `"_blank"`) anzeigt, wird im nächsten Kapitel vorgestellt. Es muss sich im gleichen Verzeichnis befinden wie die aus dem vorliegenden Formularfilm erstellte SWF-Datei.

Im Bild *danke* brauchen Sie auf der Ebene *actions* nur noch die folgende kurze Anweisung einzufügen:

```
stop();
```

Abbildung 9-7: Die zweite Seite des Bestellformulars

In diesem Kapitel:

- Server-Technologien im Vergleich
- Eine PHP-Testumgebung einrichten
- PHP-Anwendungen und Flash-Filme
- ActionScript und XML
- Praxisbeispiel: Bestellannahme und Gästebuch

KAPITEL 10

Interaktion mit Webserver-Anwendungen

We're entering an era in which software learns from its users and all of the users are connected.

– Tim O'Reilly

In diesem Kapitel erhalten Sie einen Einstieg in die Programmierung serverseitiger Skripte, die mit Flash-Filmen interagieren. Auf diese Weise erhalten Sie die Möglichkeit, Daten aus den im vorigen Kapitel vorgestellten Formularen entgegenzunehmen, auf dem Server zu verarbeiten und anschließend wieder an andere Flash-Filme weiterzugeben.

Es gibt zahlreiche verschiedene Technologien und Programmiersprachen für serverseitige Anwendungen, sowohl kommerzielle als auch frei verfügbare. Welche sich für Sie anbietet, hängt vom Betriebssystem, von der Webserver-Software und möglicherweise vom Angebot Ihres Webhosting-Providers ab. Im ersten Abschnitt dieses Kapitels werden diverse Plattformen für Server-Anwendungen im Vergleich zueinander vorgestellt; anschließend wird die beliebte, frei verfügbare Sprache PHP vertieft. Damit Sie auf Ihrem eigenen Rechner eine Testumgebung für PHP-Anwendungen zur Verfügung haben, wird in diesem Zusammenhang auch die Einrichtung des Webservers Apache und der PHP-Sprachumgebung erläutert.

Server-Technologien im Vergleich

Von der grundlegenden Idee her funktionieren alle Technologien für Webserver-Anwendungen gleich: Der Webserver übermittelt Formulardaten, die ein Benutzer eingegeben hat, an ein separates Programm. Dieses Programm verarbeitet diese Daten und erzeugt daraus – oft mit Hilfe externer Datenquellen wie zum Beispiel einer Datenbank – Ausgabedaten, die der Webserver wieder an den Browser des Benutzers sendet. Auf diese Weise entstehen dynamisch erzeugte Dokumente, meist im HTML-Format. Trotz dieser Gemeinsamkeit bestehen bedeutende Unter-

schiede zwischen den diversen Technologien. Einige von ihnen werden in den folgenden Unterabschnitten angesprochen; PHP, eine der beliebtesten, wird in den restlichen Abschnitten dieses Kapitels genauer vorgestellt.

Was die Zusammenarbeit mit Flash angeht, funktionieren Server-Anwendungen übrigens ein wenig anders: Normalerweise geht es nicht darum, dass der Server aus dynamischen Daten vollständige SWF-Dateien erzeugt.[1] Stattdessen funktioniert die Kommunikation in der Regel folgendermaßen: Ein Flash-Film stellt der Server-Anwendung Daten über die im vorigen Kapitel besprochenen Formulare zur Verfügung; bei der Dateneingabe gibt es also keinen prinzipiellen Unterschied zu Webanwendungen mit HTML-Benutzeroberfläche. Die Daten, die die Anwendung generiert, werden allerdings an einen bereits laufenden Flash-Film geliefert. Dazu dienen die bereits in Kapitel 8 vorgestellten Anweisungen `loadVariablesNum()` beziehungsweise `loadVariables()` – nur, dass in diesem Fall keine statische Textdatei, sondern eine Server-Anwendung als Quell-URL dient.

Neben dieser klassischen Form der Webanwendung gibt es inzwischen einige modernere Formen. Fast alle verwenden zur Kommunikation XML-Datenformate, die auf unterschiedliche Weise verarbeitet werden können. Besonders starkes Wachstum verzeichnen hier die so genannten Web Services. Dies sind Programmkomponenten, die das Web-Protokoll HTTP zum Austausch von XML-Steuerdaten benutzen; das konkrete XML-Format der Web Services heißt SOAP (Simple Object Access Protocol). Der Vorteil besteht darin, dass eine allgemein verfügbare Kommunikationsschnittstelle benutzt wird, so dass es keine Rolle mehr spielt, welches Betriebssystem oder welche Programmiersprache die beteiligten Anwendungen benutzen. Flash 8, besonders die Professional-Variante, enthält bereits eingebaute Unterstützung für die XML-Datenkommunikation sowie für den SOAP-Standard. Einen Einstieg in das Thema XML bietet das vorliegende Kapitel; Beispiele zu Web Services und einige Links finden Sie auf der Website zum Buch (*http://buecher.lingoworld.de/flash8/*).

CGI

Das Common Gateway Interface (CGI) ist die klassische Methode für die Zusammenarbeit zwischen Webserver und Anwendungen. Es handelt sich nicht um eine bestimmte Programmiersprache, sondern lediglich um einen seit 1995 unverändert gültigen Kommunikationsstandard: Ein CGI-Programm liest Formulardaten über Standardschnittstellen des Betriebssystems, als würden sie über die Tastatur eingetippt. Seine Standardausgabe, die normalerweise auf dem Bildschirm landen würde, leitet der Webserver als dynamisch generiertes Dokument an den Browser des

1 Prinzipiell ist dies zwar möglich, weil Macromedia das SWF-Format offen gelegt hat und weil es für viele Programmiersprachen entsprechende Schnittstellen gibt. Hier geht es aber um die Kommunikation zwischen Server-Anwendungen und auf die klassische Weise erzeugten Flash-Filmen.

Benutzers weiter. CGI arbeitet langsamer als andere Server-Programmierschnittstellen, wird dafür aber von fast jedem Webserver unterstützt und kann in beliebigen Sprachen programmiert werden. Die beliebteste Sprache für die CGI-Programmierung ist Perl – vor allem, weil sie über hervorragende Möglichkeiten zur Textmanipulation verfügt und weil ein Standardmodul mit speziellen CGI-Funktionen zur Verfügung steht.

PHP

Die Programmiersprache PHP ist zurzeit die beliebteste und verbreitetste Lösung für Webserver-Anwendungen. Sie ist frei verfügbar, arbeitet mit vielen verschiedenen Webservern zusammen und läuft auf zahllosen UNIX-Varianten sowie unter Windows; die aktuelle Version ist 5.1. PHP besticht durch eine leicht zu erlernende, an C und Perl angelehnte Syntax, unzählige Spezialfunktionen für die Web-Programmierung sowie Schnittstellen zu den meisten bekannten Datenbanksystemen. Dabei wird PHP unmittelbar an die passenden Stellen von HTML-Dokumenten geschrieben. Wenn Sie eine gründliche Einführung in die Sprache brauchen, empfehle ich Ihnen das Buch *PHP 5 – Ein praktischer Einstieg* von Ulrich Günther, das in derselben Reihe beim O'Reilly Verlag erschienen ist wie der vorliegende Band.

In den folgenden Abschnitten dieses Kapitels erfahren Sie zunächst, wie Sie Apache und PHP auf Ihrem Rechner einrichten können. Anschließend wird erläutert, wie Ihre Flash-Filme mit PHP-Anwendungen zusammenarbeiten können.

Java Servlets und JSP

Die Programmiersprache Java wurde 1995 von Sun Microsystems eingeführt. Im Gegensatz zu Sprachen wie C oder C++ braucht ein Java-Programm nur einmal kompiliert (übersetzt) zu werden, weil es nicht direkt vom Prozessor eines konkreten Rechners, sondern von einer so genannten virtuellen Maschine ausgeführt wird, die als Software für zahlreiche Plattformen und Betriebssysteme sowie für Webbrowser verfügbar ist. Die beliebteste Anwendung von Java waren ursprünglich so genannte Applets – kleine Programme, die direkt in Webseiten eingebettet und von der virtuellen Maschine des Browsers ausgeführt wurden. Flash und ähnliche Technologien haben die Bedeutung von Applets allerdings stark zurückgehen lassen; das wichtigste Einsatzgebiet von Java ist heutzutage die Programmierung von Enterprise-Anwendungen, also verteilten, datenbankgestützten Netzwerkanwendungen für (meist größere) Unternehmen.

Für Webserver-Anwendungen gibt es gleich zwei Java-Technologien: Java Servlets sind eigenständige Java-Programme, die eine Webseite als Ausgabe erzeugen – sie eignen sich also vor allem für arbeitsintensive Anwendungen mit wenig Design. Java Server Pages (JSP) sind dagegen HTML-Dokumente, in denen Java-Codeblöcke untergebracht werden können. Dieses Modell ähnelt PHP oder ASP und ist

praktischer für Seiten mit viel statischem Inhalt und wenigen programmierten Elementen. Die beliebteste Serversoftware für Servlets und JSP ist Apache Tomcat.

ASP und ASP.NET

Microsoft stattet seine Server-Betriebssysteme (Windows NT Server 4.0, Windows 2000 Server und Windows Server 2003) seit Jahren mit einem eigenen Webserver aus, dem Internet Information Server (IIS). Dieser unterstützt neben CGI eine eigene Schnittstelle für Webanwendungen: Active Server Pages (ASP) werden wie PHP oder JSP unmittelbar in den HTML-Code hineingeschrieben. Die Programmiersprache der ursprünglichen ASP war ausschließlich VBScript, eine abgespeckte Variante von Visual Basic. Die neuere Version ASP.NET, die im Rahmen von Microsofts .NET-Initiative entwickelt wurde, versteht dagegen sämtliche Sprachen der .NET Common Language Runtime (CLR) – zurzeit sind dies Visual Basic, Visual C++ sowie Visual C#. Komfortable Schnittstellen für Datenbankzugriffe, XML-Anwendungen oder Web Services machen ASP.NET zur ersten Wahl für Webanwendungen in einer Microsoft-basierten IT-Landschaft.

Macromedia ColdFusion

Der Anwendungsserver ColdFusion wurde von der Firma Allaire entwickelt, die vor einigen Jahren von Macromedia übernommen wurde. Der besondere Vorteil von ColdFusion ist die leicht zu erlernende Sprache CFML (ColdFusion Markup Language), die HTML unmittelbar um Server-Anwendungs-Tags erweitert. ColdFusion verfügt bereits ab Werk über eine definierte Schnittstelle zu Flash-Filmen.

Flash Remoting

Macromedia Flash Remoting MX verbindet Flash mit einem Enterprise Application Server, wobei es eine Variante für J2EE (Java) und eine für Microsoft .NET gibt. Daneben existieren Drittanbieter-Lösungen für die Zusammenarbeit mit Perl- und PHP-Anwendungen. Auf diese Weise werden dynamisch generierte Flash-Filme zur Benutzeroberfläche für Enterprise-Webanwendungen. Flash Remoting erweitert ActionScript um zahlreiche Funktionen zur Kommunikation mit Server-Anwendungen. Ein weiterer Schwerpunkt ist die Unterstützung von SOAP-basierten Web Services.

Eine PHP-Testumgebung einrichten

Im nächsten Abschnitt lernen Sie einige Grundlagen der Programmiersprache PHP kennen und erfahren, wie Sie Flash-Anwendungen erstellen können, die mit PHP-Programmen kommunizieren. Um diese Technologie auf einer realen Website ein-

zusetzen, benötigen Sie einen Hosting-Dienst, der PHP-Unterstützung anbietet. Bei großen deutschen Webhostern wie 1&1 oder Strato ist diese bereits in den mittleren Tarifen enthalten. Aber sicherlich möchten Sie die hier vorgestellten (und Ihre eigenen) Anwendungen auch auf Ihrem Rechner testen, bevor Sie sie veröffentlichen. Deshalb wird in diesem Abschnitt beschrieben, wie Sie eine Testumgebung für PHP-Anwendungen einrichten können. Dazu werden drei Softwarekomponenten benötigt:

- *Ein Webserver:* PHP arbeitet mit so gut wie allen gängigen Webservern zusammen. Hier wird die Installation und Inbetriebnahme von Apache beschrieben, weil es sich um freie Software handelt und weil er der beliebteste und am weitesten verbreitete Webserver überhaupt ist. In diesem Abschnitt wird in Kürze das Wichtigste über Apache erläutert.
- *PHP:* Die Sprache PHP ist ebenfalls Open Source-Software. Sie wird für zahlreiche Plattformen und Betriebssysteme angeboten. In diesem Abschnitt wird die Einrichtung der neuesten Version (PHP 5) unter Windows und Mac OS X in Zusammenarbeit mit dem Webserver Apache beschrieben.
- *Eine Datenbank:* Zwar benötigt nicht jede Webanwendung Zugriff auf ein Datenbanksystem, aber sobald mehr als einige wenige Daten zusammenkommen, gibt es keine effizientere Speichermöglichkeit. Deshalb setzen die meisten Produktkataloge, Foren oder Gästebücher auf eine Datenbank auf. PHP kann auf beinahe jede existierende Datenbank zugreifen. Die häufigste Wahl ist allerdings das freie Datenbanksystem MySQL, so dass diese Kombination hier beschrieben wird.



Wenn Sie möchten, können Sie die in den nachfolgenden Abschnitten beschriebenen Schritte deutlich abkürzen: Unter *http://www.apachefriends.org/de* können Sie sich das freie Paket XAMPP herunterladen. Es handelt sich um ein sehr leicht zu installierendes, vorkonfiguriertes Bündel der neuesten Versionen von Apache, PHP, MySQL und weiterer interessanter (Server-)Software, das für Windows, Linux und Sun Solaris angeboten wird.

Apache installieren und konfigurieren

Der Webserver Apache wird nach wie vor in den Versionen 1.3 und 2.x gepflegt, wobei nur Apache 2 um neue Funktionen ergänzt wird. Während bei vielen Hosting-Diensten noch die Version 1.3 installiert ist, sollten Sie für eine Neuinstallation die modernere und leistungsfähigere Version 2 verwenden. Sie finden die aktuelle Version 2 auf der CD zum Buch, und zwar in zwei Varianten: Installer-Paket Version 2.0.55 für Windows und Quellcode-Paket Version 2.2.0 für UNIX-Systeme (zum Beispiel Mac OS X oder Linux).

In diesem Unterabschnitt wird in Kürze beschrieben, wie Sie Apache 2 auf Ihrem System installieren und für die (lokale) Auslieferung von HTML-Dokumenten und PHP-Anwendungen konfigurieren können. Weitere Informationen erhalten Sie in der umfangreichen Online-Dokumentation (*http://httpd.apache.org/docs-2.2/*), die standardmäßig auch mit Apache selbst auf Ihrem System installiert wird. Darüber hinaus kann ich Ihnen mein Buch *Apache 2* (2. Auflage, Bonn 2006, Galileo Press) empfehlen; auf der zugehörigen Website (*http://buecher.lingoworld.de/apache2*) finden Sie ebenfalls sehr viel Dokumentationsmaterial.

Apache unter UNIX kompilieren

Auf das Kompilieren des Quellcode-Pakets (*httpd-2.2.0.tar.gz*) wird hier nicht allzu genau eingegangen. Unter dem Flash-relevanten Betriebssystem Mac OS X ist es in der Regel nicht erforderlich, weil Apache standardmäßig automatisch mit dem System installiert wird. Wenn Sie Ihren Webserver dagegen auf einem Linux- oder sonstigen UNIX-System betreiben möchten, können Sie der nachfolgenden Kurzanleitung folgen.

Zunächst müssen Sie die Archivdatei entpacken und anschließend in das entsprechende Verzeichnis wechseln:

```
# tar -xzvf httpd-2.2.0.tar.gz
# cd httpd-2.2.0
```

Vor dem eigentlichen Kompiliervorgang wird nun das Konfigurationsskript aufgerufen, das die Makefiles an Ihr Betriebssystem und Ihre Sonderwünsche anpasst. Geben Sie für Standardoptionen einfach Folgendes ein:

```
# ./configure
```

So wird Apache 2.2 zur Installation mit den wichtigsten Modulen unter dem Verzeichnis */usr/local/apache2* konfiguriert. Alternative Optionen können Sie – vor der Ausführung des eigentlichen Konfigurationsbefehls – folgendermaßen erfahren:

```
# ./configure --help |less
```

Nun müssen Sie noch (unbedingt als `root`!) die Standardbefehle zur Kompilierung und Installation eingeben:

```
# make
# make install
```

Anschließend können Sie den Server folgendermaßen in Betrieb nehmen:

```
# /usr/local/apache2/bin/apachectl start
```

Falls Sie sich bei der Konfiguration für ein anderes Installationsverzeichnis entschieden haben, gilt dies hier sinngemäß.

Apache unter Windows installieren

Für Windows-Systeme stellt die Apache Group einen komfortablen Installer bereit. Sie finden das Paket *apache_2.0.55-win32-x86-no_ssl.msi* auf der CD zum Buch. Es handelt sich um eine Windows-Installer-Datei. Wenn Ihr System älter als Windows 2000 beziehungsweise Windows ME ist, müssen Sie zunächst den Windows Installer selbst installieren; auch dieser ist auf der beiliegenden CD-ROM zu finden.

Doppelklicken Sie zur Installation auf die *.msi*-Datei. Anschließend müssen Sie auf mehreren Dialogseiten die Fragen des Installers beantworten (und jeweils die Schaltfläche *Next* betätigen):

1. Bestätigung, dass Apache installiert werden soll.
2. Zustimmung zur Apache-Lizenz (*I accept the terms in the license agreement*) – es handelt sich um eine typische Open Source-Lizenz, die Anwendern beinahe völlige Freiheiten lässt. Ihr Sinn besteht vielmehr darin, die freie Software vor einer Vereinnahmung durch kommerzielle Unternehmen oder gar Softwarepatente zu schützen.
3. Einführende Informationen über den Apache-Webserver mit einer Liste weiterführender Informationsquellen.
4. Grundeinstellungen für den Server-Betrieb: *Network Domain* ist der Domain-Name Ihres Netzes. Wenn Apache nur lokal ausgeführt werden soll, geben Sie so etwas wie *test.local* ein. *Server Name* ist der Name, unter dem der Webserver im Netzwerk erreichbar sein soll – in der Regel *www.[Ihr Domain-Name]*, also zum Beispiel *www.test.local*. Beachten Sie, dass diese Einstellungen lediglich für die Anpassung der Apache-Konfigurationsdatei genutzt werden; die Erreichbarkeit des Servers über diesen Namen wird dadurch nicht gewährleistet. Dazu müssen Sie den Server-Namen vielmehr in die Datei *[Windows-Verzeichnis]\System32\drivers\etc\hosts* eintragen.[2] Beispiel:

   ```
   192.168.0.5 www.test.local
   127.0.0.1   www.test.local
   ```

   Statt *192.168.0.5* müssen Sie die Netzwerkadresse (IP-Adresse) Ihres Rechners angeben; *127.0.0.1* ist dagegen die so genannte Loopback-Adresse, über die ein Rechner Netzwerkkommunikation mit sich selbst betreiben kann. Wenn der Webserver auch im lokalen Netzwerk über den angegebenen Namen ansprechbar sein soll, müssen Sie die erste der beiden Zeilen auch in den *hosts*-Dateien der anderen Rechner eintragen; alternativ können Sie auch einen entsprechenden Konfigurationseintrag auf einem Nameserver vornehmen.

   Die Angabe unter *Administrator's Email Address* wird verwendet, um auf automatisch generierten Fehlermeldungsseiten einen E-Mail-Link unterzubringen, damit Besucher den Server-Betreiber über eventuelle Probleme informieren

---

2 Unter UNIX ist dies ebenfalls der Fall. Die entsprechende Datei ist */etc/hosts*.

können. Normalerweise wird hier eine E-Mail-Adresse in der Form *webmaster@[Ihre Domain]* verwendet, gemäß dem obigen Beispiel also *webmaster@test.local*.

In aller Regel sollten Sie ganz unten die Option *for All Users, on Port 80, as a Service* aktivieren. Dies sorgt dafür, dass Apache beim Hochfahren von Windows automatisch als regulärer Webserver gestartet wird.

5. Installationstyp: Für gewöhnlich sollten Sie hier *Custom* wählen, damit möglichst alles installiert werden kann.
6. Wenn Sie im vorigen Schritt *Custom* gewählt haben, können Sie nun die Komponenten auswählen, die Sie installieren möchten. Wenn Sie sich keine Sorge um Festplattenplatz machen müssen, sollten Sie hier einfach unter *Apache HTTP Server 2.0.55* den Punkt *This Feature, and all subfeatures, will be installed on local hard drive* aus dem Pop-up-Menü auswählen. Unten auf derselben Registerkarte können Sie das Verzeichnis wählen, in das Apache installiert werden soll – der Vorgabewert ist *[Windows-Laufwerk]\Programme\Apache Group*.
7. Auf der letzten Seite müssen Sie sich endgültig entscheiden: Klicken Sie auf *Install*, um die Installation mit den gewählten Optionen durchzuführen, oder auf *Back*, um sie noch einmal zu ändern.

Nach der Installation können Sie Apache über den *Apache-Monitor*, das kleine Feder-Symbol im Systray (Bereich der Taskleiste neben der Uhrzeit), steuern.

Grundkonfiguration und Test

Die Konfiguration des Apache-Webservers erfolgt nicht über eine grafische Oberfläche, sondern über eine zentrale Konfigurationsdatei namens *httpd.conf*, die sich im Verzeichnis *conf* Ihrer Apache-Installation befindet. Sie können sie mit einem beliebigen Texteditor öffnen und bearbeiten. Nach jeder Änderung müssen Sie Apache neu starten (siehe unten).

Je nach installierten Modulen gibt es über 400 verschiedene Konfigurationsanweisungen (Direktiven); die meisten von ihnen sind allerdings für Spezialaufgaben zuständig und können in der bereits erwähnten Online-Dokumentation oder in Fachbüchern nachgeschlagen werden. Erfreulicherweise werden bei der Installation fast alle Voreinstellungen korrekt vorgenommen. Einige wichtige Direktiven sollten Sie aber trotzdem kontrollieren:

- `ServerName` ist der Name des Webservers; die Bedeutung dieser Einstellung wurde bereits weiter oben im Rahmen der Installation unter Windows erläutert. Hier ein Beispiel:

  ```
  ServerName www.test.local
  ```

- `Listen` gibt die Netzwerkadresse und den TCP-Port für den Server an. Bei einem Standard-Webserver wird einfach Port 80 ohne Adresse angegeben,

damit er an Port 80 aller Netzwerkkarten und -verbindungen auf Anfragen lauscht. Der entsprechende Konfigurationseintrag sieht so aus:

```
Listen 80
```

Wenn Apache zusätzliche Aufgaben erfüllt, beispielsweise als Proxy-Server oder als sicherer HTTPS-Server, benötigen Sie dafür weitere `Listen`-Zeilen.

- `ServerRoot` gibt das Apache-Stammverzeichnis an, unter dem sich Verzeichnisse wie *conf* (Konfigurationsdateien) oder *logs* (Protokolldateien) befinden. Normalerweise sollte bei der Installation der korrekte Wert eingetragen worden sein. Unter UNIX sieht der Eintrag normalerweise so aus:

  ```
  ServerRoot /usr/local/apache2
  ```

  Auf Windows-Systemen hängt die Einstellung vom gewählten Installationsverzeichnis ab. Beispiel:

  ```
  ServerRoot "C:/Programme/Apache Group/Apache2"
  ```

  Beachten Sie, dass die Anführungszeichen erforderlich sind, wenn der Pfadname ein Leerzeichen enthält. Außerdem müssen Sie in der Apache-Konfigurationsdatei auch unter Windows den Slash (/) als Pfad-Trennzeichen benutzen.

- `DocumentRoot` gibt das Stammverzeichnis der Website an – die Startseite aus diesem Verzeichnis wird an Besucher ausgeliefert, wenn diese den einfachen Servernamen als URL anfordern. Der Wert kann ein absolutes Verzeichnis sein, andernfalls gilt er relativ zur `ServerRoot`. Beispiele:

  ```
  DocumentRoot /usr/local/apache2/htdocs
  DocumentRoot "C:/Programme/Apache Group/Apache2/htdocs"
  ```

- `DirectoryIndex` gibt den Namen der Startseite an, nach der Apache suchen soll, wenn ein Verzeichnis angefordert wird. Der Standardwert ist `index.html`. Wenn Sie möchten, können Sie auch eine Liste mehrerer Dateien verwenden, nach denen in der angegebenen Reihenfolge gesucht wird. Das folgende Beispiel liefert die erste gefundene Datei aus der Reihe `index.html`, `index.htm` oder `index.php` aus:

  ```
  DirectoryIndex index.html index.htm index.php
  ```

Als Nächstes sollten Sie sehr restriktive Einstellungen für alle Verzeichnisse auf dem Server-Rechner vornehmen, damit niemand auf unerlaubte Bereiche zugreifen kann. Setzen Sie dazu die folgenden Verzeichnisoptionen für das Wurzelverzeichnis `/`:

```
<Directory />
  Options None
  AllowOverride None
  Order deny,allow
  Deny from all
</Directory>
```

Zugriffe auf die `DocumentRoot` sollen dagegen gestattet sein. Wenn Sie nur von Ihrem eigenen Rechner aus darauf zugreifen möchten, können Sie Folgendes schreiben (wobei Sie den Pfad natürlich gemäß Ihrer `DocumentRoot` anpassen müssen):

```
<Directory /usr/local/apache2/htdocs>
  Options Indexes FollowSymLinks
  AllowOverride None
  Order deny,allow
  Deny from all
  Allow from 127.0.0.1
</Directory>
```

Sollen dagegen auch Rechner aus Ihrem lokalen Netzwerk Zugriff erhalten, dann müssen Sie die `Allow`-Zeile um die gemeinsame Netzwerknummer ergänzen. Beispiel:

```
Allow from 127.0.0.1 192.168.0
```

Was Sie hier statt *192.168.0* eintragen müssen, hängt von Ihrer Netzwerkkonfiguration ab. Für einen öffentlich im Internet zugänglichen Webserver werden dagegen folgende Zeilen benutzt:

```
Order allow,deny
Allow from all
```

Nachdem Sie die entsprechenden Änderungen vorgenommen und die Konfigurationsdatei gespeichert haben, müssen Sie Apache neu starten. Unter UNIX können Sie dazu folgenden Befehl eingeben:

```
# apachectl restart
```

In welchem Verzeichnis sich das Tool *apachectl* befindet, wurde bereits angesprochen. Auf einem Windows-Rechner genügt es, das Icon des Apache-Monitors anzuklicken und *Apache 2* → *Restart* auszuwählen.

Öffnen Sie nun einen Browser auf dem Webserver-Rechner, und geben Sie die Adresse *http://127.0.0.1/* ein. Wenn alles in Ordnung ist, sollten Sie nun die automatisch generierte Apache-Startseite zu Gesicht bekommen. Anschließend können Sie auch eigene HTML-Dateien unterhalb Ihrer `DocumentRoot` speichern, um sie sich von Apache ausliefern zu lassen. Im nächsten Schritt erfahren Sie nun, wie Sie PHP-Unterstützung hinzufügen können.

MySQL installieren

Der freie Datenbankserver MySQL ist inzwischen eines der beliebtesten Datenbanksysteme, insbesondere für Webanwendungen. Damit PHP-Anwendungen problemlos mit MySQL arbeiten können, sollten Sie MySQL vor PHP installieren. In diesem Abschnitt wird die Installation der aktuellen stabilen Version 5.0 für Windows- und UNIX-Systeme beschrieben.

Installation auf UNIX-Systemen

Auf der CD-ROM zum Buch sind MySQL 5.0-Binärpakete für Mac OS X (10.3 Jaguar und 10.4 Tiger) sowie Linux (Intel x86) enthalten; auf der MySQL-Website *http://www.mysql.com* finden Sie weitere Pakete für viele verschiedene UNIX-Vari-

anten. Wenn für Sie nichts Passendes dabei ist, müssen Sie selbst das Quellcodepaket kompilieren; es eignet sich zur Installation auf jedem beliebigen UNIX-System und befindet sich ebenfalls auf der beiliegenden CD.

Das Mac OS X-Paket ist ein bequemer Installer. Bevor Sie ihn ausführen, müssen Sie unter *Systemeinstellungen* → *Benutzer* überprüfen, ob bereits ein User namens `mysql` existiert, und diesen ansonsten neu anlegen. Anschließend können Sie die Image-Datei (*.dmg*) per Doppelklick mounten und den enthaltenen Installer durch einen weiteren Doppelklick starten. MySQL wird unter */usr/local/mysql-Version* installiert; zudem erstellt der Installer den Symlink */usr/local/mysql*. Anschließend wird automatisch das Skript *mysql_install_db* zur Erstellung der Verwaltungstabellen ausgeführt.

Bei der Linux-Binärvariante ist etwas mehr Handarbeit erforderlich. Vor der Installation sollten Sie auch hier einen Benutzer und eine Gruppe namens *mysql* einrichten, damit MySQL aus Sicherheitsgründen unter diesen IDs ausgeführt wird. Unter Linux und vielen anderen UNIX-Systemen funktioniert dies mit den folgenden beiden Befehlen (als `root`):

```
# groupadd mysql
# useradd -g mysql mysql
```

Anschließend wird das Binärpaket entpackt. Beispiel:

```
# tar xzvf mysql-max-5.0.18-linux-i686-glibc23.tar.gz
```

Verschieben Sie das entpackte Verzeichnis nach */usr/local*, und erstellen Sie einen symbolischen Link darauf, der einfach *mysql* heißt:

```
# mv mysql-max-5.0.18-linux-i686-glibc23 /usr/local
# ln -s /usr/local/mysql-max-5.0.18-linux-i686-glibc23 /usr/local/mysql
```

Danach können Sie in das Verzeichnis */usr/local/mysql* wechseln und müssen dort das Skript *mysql_install_db* im Unterverzeichnis *bin* aufrufen, das die Datenbank `mysql` mit den Tabellen für Benutzerrechte und weitere Verwaltungsaufgaben einrichtet:

```
# cd /usr/local/mysql
# bin/mysql_install_db
```

Als Nächstes müssen einige Benutzer- und Gruppenrechte angepasst werden – die Dateien in allen MySQL-Unterverzeichnissen sollen dem User *root* und der Gruppe *mysql* gehören, das Unterverzeichnis `data` mit den Datenbanken dagegen auch dem Benutzer *mysql*. Geben Sie dazu folgende Anweisungen ein:

```
# chown -R root .
# chown -R mysql data
# chgrp -R mysql .
```

Anschließend können Sie den Datenbank-Server starten:

```
# /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

Wenn MySQL beim Booten automatisch gestartet werden soll, müssen Sie in Ihrem Verzeichnis für Startskripte (unter vielen Linux-Distributionen beispielsweise */etc/init.d*) einen symbolischen Link auf das Skript *support_files/mysql.server* aus dem MySQL-Verzeichnis erstellen. Beispiel:

```
# ln -s /usr/local/mysql/support_files/mysql.server /etc/init.d/mysql
```

Anschließend muss dieses Skript auf eine jeweils betriebssystemspezifische Art und Weise aktiviert werden. SUSE und einige andere Linux-Distributionen bieten dafür zum Beispiel das bequeme Kommando `chkconfig` an:

```
# chkconfig -a mysql
```

Unter Linux, Mac OS X und allen anderen UNIX-Systemen können Sie alternativ auch das Quellcode-Paket kompilieren. Entpacken Sie es, und wechseln Sie in das neue Verzeichnis:

```
# tar -xzvf mysql-5.0.18.tar.gz
# cd mysql-5.0.18
```

Geben Sie nun die üblichen Befehle zur Konfiguration, Kompilierung und Installation ein. Für das Verzeichnis */usr/local/mysql* und die User-ID *mysql* sehen diese so aus:

```
# ./configure --prefix=/usr/local/mysql --with-mysqld-user=mysql
# make
# make install
```

Anschließend müssen Sie *mysql_install_db* ausführen und auch die nachfolgenden Schritte ausführen, die oben für die Linux-Binärinstallation beschrieben wurden.

Nach Installation und Start des MySQL-Servers können Sie zum Beispiel den Kommandozeilen-Client `mysql` verwenden, um Datenbanken einzurichten und zu verwalten. Es befindet sich im *bin*-Verzeichnis der Installation und kann beim ersten Start ohne Passwort ausgeführt werden:

```
# /usr/local/mysql/bin/mysql
```

Die neue Eingabeaufforderung `mysql>` zeigt, dass Sie sich nun innerhalb dieses Clients befinden. Hier sollten Sie sich aus Sicherheitsgründen zuerst darum kümmern, ein Passwort für `root@localhost` – die Administration vom lokalen Rechner aus – festzulegen:

```
mysql> SET PASSWORD FOR root@localhost = PASSWORD("meinPasswort");
```

Anschließend sollten Sie alle Zugangsberechtigungen mit leerem Benutzernamen und/oder leerem Passwort entfernen:

```
mysql> DELETE FROM mysql.user WHERE user="" OR password="";
```

Nun können Sie den Client zunächst wieder verlassen, indem Sie `exit` oder `\q` eingeben. Wenn Sie ihn das nächste Mal starten, müssen Sie die Optionen `-u` *`Benutzername`* und `-p` (Passworteingabeaufforderung) verwenden:

```
$ mysql -u root -p
[Passwort eingeben]
```

Installation unter Windows

Für Windows bieten die MySQL-Entwickler einen bequemen Binär-Installer an. Entpacken Sie als Erstes die ZIP-Datei *mysql-5.0.18-win32.zip*. Anschließend können Sie die enthaltene Datei *Setup.exe* per Doppelklick ausführen. Falls auf Ihrem Rechner bereits eine ältere MySQL-Version existiert, wird sie deinstalliert und durch die neue ersetzt; die vorhandenen Datenbanken bleiben erhalten. Als Erstes müssen Sie sich den Installationsumfang aussuchen: *Typical* (die wichtigsten Komponenten), *Complete* (alles) oder *Custom* (freie Auswahl). Wählen Sie Letzteres, und klicken Sie auf dem nächsten Bildschirm auf die Schaltfläche *Change*, wenn Sie das Installationsverzeichnis wechseln möchten. Ein Klick auf *Next* startet anschließend die Installation.

Nach der Installation können Sie *Configure the MySQL Server now* wählen, um den Datenbankserver nach Ihren Wünschen einzurichten – beispielsweise wird er als automatisch startender Dienst installiert. Der Konfigurationsdialog ist später jederzeit über das Startmenü verfügbar. Beantworten Sie die Fragen für einen Programmierrechner wie folgt:

1. Wählen Sie den Installationstyp *Detailed Configuration*.
2. Entscheiden Sie sich für den Servertyp *Developer Machine* – MySQL wird nicht mit Priorität ausgeführt, sondern – wie auf einem Arbeitsrechner nötig – als ein Programm unter vielen.
3. Als Datenbanktyp sollten Sie *Multifunctional Database* wählen, um flexibel mit allen MySQL-Tabellentypen arbeiten zu können.
4. Unter *Tablespace* können Sie Laufwerk und Verzeichnis für Ihre MySQL-Datenbanken einstellen.
5. Wählen Sie *Decision Support* für maximal 20 gleichzeitige Client-Verbindungen – mehr braucht ein Testrechner nicht.
6. Behalten Sie den Standard-TCP-Port 3306 bei; auch *Enable Strict Mode* für eine strengere Datenbankabfrage-Syntax sollten Sie eingeschaltet lassen.
7. Die beste Zeichensatzeinstellung für Webanwendungen ist *Standard Character Set* (ISO-8859-1).
8. Wählen Sie *Install As Windows Service* und *Launch the MySQL Server automatically*, um MySQL als Dienst zu installieren und beim Booten automatisch zu starten. *Include Bin Directory in Windows PATH* sorgt – nach einem Systemneustart – dafür, dass Sie die MySQL-Kommandozeilen-Hilfsprogramme aus jedem Verzeichnis aufrufen können.
9. Aktivieren Sie *Modify Security Settings*, und geben Sie ein Passwort für den MySQL-Administrator `root` ein. *Enable root access from remote machines* sollte deaktiviert werden – die Fernadministration von Servern sollte ein weniger privilegierter Benutzer vornehmen. *Create An Anonymous Account* ist in der Regel ebenfalls nicht empfehlenswert.

10. Wenn Sie auf *Execute* klicken, werden die gewählten Änderungen durchgeführt. Falls es nicht klappen sollte, müssen Sie überprüfen, ob eventuell bereits ein älterer MySQL-Dienst läuft oder ob Ihre lokale Firewall den Port 3306 blockiert.

Über MySQL

Wenn Sie eine gründliche Einführung in MySQL benötigen, können Sie sich mein Buch *Praktischer Einstieg in MySQL mit PHP* aus der vorliegenden O'Reilly-Buchreihe besorgen. Hier finden Sie einige Informationen für die ersten Schritte.

Wie die meisten Datenbankserver ist MySQL eine so genannte *relationale Datenbank*. Dieser Begriff besagt, dass die Daten in Tabellen organisiert sind, die zueinander in Beziehung (Relation) gesetzt werden können. Dies sorgt dafür, dass Daten niemals doppelt in der Datenbank gespeichert werden müssen. Denken Sie zum Beispiel an eine CD-Datenbank: Eine Tabelle enthält Informationen über die einzelnen CDs wie Interpret, Titel, Spielzeit und so weiter. Da mehrere CDs vom selben Interpreten stammen können, empfiehlt es sich, die Interpreten in einer separaten Tabelle zu speichern. Der Eintrag Interpret in der CD-Tabelle ist dann lediglich die Nummer des entsprechenden Interpreten aus der Interpreten-Tabelle.

Für die Arbeit mit relationalen Datenbanken sollten Sie einige Begriffe kennen: Eine Tabellenzelle, die eine Information über einen einzelnen Gegenstand enthält, heißt *Datenfeld*. Eine Zeile mit sämtlichen Informationen über einen Gegenstand wird *Datensatz* genannt (englisch *Record*). Wenn Sie aus einer Tabelle heraus auf Datensätze einer anderen Tabelle verweisen möchten, benötigen diese Datensätze je ein Feld mit einem einmaligen Wert. Dieses spezielle Feld heißt *Primärschlüssel*. Der Primärschlüssel wird oft durch einfaches Durchnummerieren gebildet. Einige Objekte besitzen dagegen eine Art »natürlichen« Primärschlüssel: Bei Autos ist dies etwa das amtliche Kennzeichen, bei Büchern dagegen die ISBN.

Fast alle relationalen Datenbanken können über eine Sprache namens SQL (Structured Query Language) gesteuert werden – MySQL leitet sogar seinen Namen von dieser Tatsache ab. Es handelt sich um eine leicht zu erlernende, mächtige Sprache, in der so genannte Abfragen formuliert werden. Damit können Sie Tabellen erstellen, ändern und vor allem Informationen daraus erhalten.

Erste Schritte mit MySQL

Selbstverständlich würde eine systematische Einführung in MySQL in diesem Kapitel zu weit führen. Hier finden Sie nur ein kurzes Beispiel, das Sie direkt eintippen können, wenn Sie den oben erwähnten `mysql`-Client starten:

```
$ mysql -u root -p
[Passwort eingeben]
```

Das Beispiel erstellt die Datenbank musik mit zwei Tabellen: interpreten mit den Feldern int_nr und int_name sowie cds mit den Feldern cd_nr, cd_interpret, cd_titel, cd_jahr und cd_songs (Anzahl der Songs). Geben Sie dazu Folgendes ein:

```
mysql> CREATE DATABASE musik;
Query OK, 1 row affected (0.24 sec)
mysql> use musik
Database changed
mysql> CREATE TABLE interpreten
    -> (int_nr INT AUTO_INCREMENT PRIMARY KEY, int_name VARCHAR(40));
Query OK, 0 rows affected (0.28 sec)
mysql> CREATE TABLE cds (cd_nr INT AUTO_INCREMENT PRIMARY KEY,
    -> cd_interpret INT, cd_titel VARCHAR(40), cd_jahr YEAR,
    -> cd_songs INT);
Query OK, 0 rows affected (0.07 sec)
```

Wie Sie sehen, besitzt jedes Feld einen festgelegten Datentyp. Hier werden INT (Ganzzahl), VARCHAR(n) (Text mit maximal n Zeichen Länge) und YEAR (Jahreszahl) verwendet. int_nr beziehungsweise cd_nr sind die Primärschlüssel (PRIMARY KEY) der beiden Tabellen; der Zusatz AUTO_INCREMENT besagt, dass sie automatisch durchnummeriert werden sollen.

Fügen Sie nun mit den beiden folgenden Zeilen Beispielwerte in die Tabelle interpreten ein:

```
mysql> INSERT INTO interpreten (int_name) VALUES ("Metallica");
Query OK, 1 row affected (0.08 sec)
mysql> INSERT INTO interpreten (int_name) VALUES ("Extreme");
Query OK, 1 row affected (0.06 sec)
```

Nun können Sie sämtliche Inhalte der Tabelle interpreten auswählen, damit diese angezeigt wird:

```
mysql> SELECT * FROM interpreten;
+--------+-----------+
| int_nr | int_name  |
+--------+-----------+
|      1 | Metallica |
|      2 | Extreme   |
+--------+-----------+
2 rows in set (0.09 sec)
```

Dies zeigt die automatisch eingefügten Nummern an, die Sie benötigen, um den CDs die korrekten Interpreten zuzuordnen. Mit dieser Information können Sie sich daranmachen, CDs einzugeben:

```
mysql> INSERT INTO cds (cd_interpret, cd_titel, cd_jahr, cd_songs)
    -> VALUES (2, "III Sides to Every Story", 1992, 14);
Query OK, 1 row affected (0.03 sec)
mysql> INSERT INTO cds (cd_interpret, cd_titel, cd_jahr, cd_songs)
    -> VALUES (2, "Pornograffitti", 1990, 13);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO cds (cd_interpret, cd_titel, cd_jahr, cd_songs)
    -> VALUES (1, "Metallica", 1991, 12);
Query OK, 1 row affected (0.00 sec)
```

Der folgende Befehl zeigt die gesamte Tabelle `cds` an, und zwar aufsteigend (*ASCending*) nach Jahr sortiert:

```
mysql> SELECT * FROM CDS ORDER BY cd_jahr ASC;
+-------+-------------+-------------------------+------+----------+
| cd_nr | cd_interpret | cd_titel               | JAHR | cd_songs |
+-------+-------------+-------------------------+------+----------+
|     2 |           2 | Pornograffitti          | 1990 |       13 |
|     3 |           1 | Metallica               | 1991 |       12 |
|     1 |           2 | III Sides to Every Story | 1992 |      14 |
+-------+-------------+-------------------------+------+----------+
3 rows in set (0.07 sec)
```

Der wichtigste Bestandteil einer `SELECT`-Anfrage ist eine `WHERE`-Klausel, die Bedingungen dafür formuliert, welche Datensätze ausgewählt werden sollen. Das folgende Beispiel zeigt nur die beiden CDs von Extreme an:

```
mysql> SELECT * FROM cds WHERE cd_interpret=2;
+-------+-------------+-------------------------+------+----------+
| cd_nr | cd_interpret | cd_titel               | JAHR | cd_songs |
+-------+-------------+-------------------------+------+----------+
|     1 |           2 | III Sides to Every Story | 1992 |      14 |
|     2 |           2 | Pornograffitti          | 1990 |       13 |
+-------+-------------+-------------------------+------+----------+
2 rows in set (0.00 sec)
```

Zu guter Letzt sollten Sie sich noch ein Beispiel dafür anschauen, wie Relationen in der Praxis genutzt werden. Dazu soll eine Abfrage formuliert werden, die den Interpreten und den Titel jeder CD ausgibt. Die Beziehung kann entweder über eine relativ komplizierte `JOIN`-Klausel oder mit Hilfe von `WHERE` formuliert werden; hier sehen Sie Letzteres:

```
mysql> SELECT int_name, cd_titel FROM interpreten, cds
    -> WHERE int_nr=cd_interpret;
+-----------+-------------------------+
| int_name  | cd_titel                |
+-----------+-------------------------+
| Extreme   | III Sides to Every Story |
| Extreme   | Pornograffitti          |
| Metallica | Metallica               |
+-----------+-------------------------+
3 rows in set (0.08 sec)
```

Wenn mehrere Tabellen gleichnamige Felder enthalten, die Sie ansprechen möchten, müssen Sie diese übrigens in der Form *Datenbankname.Feldname* notieren. Beispiele: `cds.cd_titel` oder `interpreten.int_nr`. Das vorliegende Beispiel kommt ohne dieses Hilfsmittel aus, weil in jeder Tabelle konsequent Namenspräfixe (hier `cd_` beziehungsweise `int_`) verwendet werden. Dies ist auch in der Praxis empfehlenswert.

Für den Zugriff durch Webanwendungen sollten Sie zu guter Letzt einen separaten MySQL-Benutzer erzeugen, der nur auf die jeweilige Einzeldatenbank zugreifen darf. Das folgende Beispiel erzeugt im Kommandozeilen-Client einen Benutzer namens *dbuser*, der in Skripten auf demselben Rechner mit dem (schlechten!) Passwort `geheim` auf die Datenbank `musik` zugreifen darf:

```
mysql> CREATE USER dbuser@localhost;
mysql> GRANT ALL PRIVILEGES ON musik.* TO dbuser@localhost
    -> IDENTIFIED BY "geheim";
```

Falls Sie noch MySQL 4 verwenden, müssen Sie die `CREATE USER`-Zeile weglassen.

PHP installieren

Die Programmiersprache PHP wird seit 1995 von Rasmus Lerdorf entwickelt. Anfangs handelte es sich um eine Art Makrosammlung für Webserver-Aufgaben (*Personal Homepage Tools*); durch die Zusammenarbeit zahlreicher Entwickler hat sich das Projekt zu einer ausgereiften Server-Anwendungssprache mit unzähligen eingebauten und auch von Drittanbietern bereitgestellten Funktionen entwickelt. Sie finden die aktuelle Version 5.1 für UNIX und Windows auf der beiliegenden CD-ROM; das aktuelle Release können Sie jeweils unter *http://www.php.net* herunterladen.

In diesem Unterabschnitt wird erläutert, wie Sie PHP 5 unter UNIX und Windows installieren und für die Zusammenarbeit mit Apache 2 konfigurieren können. Beachten Sie, dass es dazu zwei Möglichkeiten gibt: PHP lässt sich als unabhängiger Interpreter installieren und kann über die CGI-Schnittstelle mit einem Webserver zusammenarbeiten. Alternativ können Sie PHP als so genanntes SAPI-Modul (Server-API) für verschiedene Webserver einrichten, was die Leistungsfähigkeit von PHP-Anwendungen erheblich verbessert. Deshalb wird hier nur die zweite Variante besprochen.

Installation unter UNIX

Auf UNIX-Systemen wie Mac OS X und Linux wird PHP bevorzugt aus den Quellcodes kompiliert, was im Wesentlichen genauso funktioniert wie bei Apache. Zunächst einmal müssen Sie das PHP-Installationspaket entpacken und in dessen Verzeichnis wechseln:

```
# tar xzvf php-5.1.2.tar.gz
# cd php-5.1.2
```

Nun müssen Sie das *configure*-Skript aufrufen. Damit PHP als Apache 2-Modul installiert wird, müssen Sie dem Skript mitteilen, wo sich bei Ihnen das Tool *apxs* befindet, das für die nachträgliche Installation von Modulen zuständig ist. Bei einer

Apache-Standardinstallation finden Sie es unter */usr/local/apache2/bin/apxs*. Weitere Konfigurationsoptionen erhalten Sie über folgende Anweisung:

```
# ./configure --help |less
```

Wenn Sie PHP als Apache-Modul mit Standardeinstellungen unter dem Verzeichnis */usr/local/php5* installieren möchten, müssen Sie nun Folgendes eingeben:

```
# ./configure --prefix=/usr/local/php5 \
--with-apxs2=/usr/local/apache2/bin/apxs
```

Nachdem *configure* seine Arbeit beendet hat, folgen die beiden üblichen Befehle zum Kompilieren und Installieren:

```
# make
# make install
```

Ihre Apache-Konfigurationsdatei sollte nun automatisch um folgende Zeile ergänzt worden sein:

```
LoadModule php5_module modules/libphp5.so
```

Sie müssen noch folgende Zeile hinzufügen, damit Dateien mit der Endung *.php* von diesem Modul verarbeitet werden:

```
AddType application/x-httpd-php .php
```

Suchen Sie dazu einfach die Stelle in *httpd.conf*, wo sich weitere `AddType`-Direktiven befinden. Auf diese Weise werden PHP-Skripte in der gesamten Website ausgeführt.

Als Nächstes müssen Sie Apache neu starten. Danach können Sie das folgende kurze Skript schreiben, um zu testen, ob PHP ordnungsgemäß installiert wurde:

```
<?php

   phpinfo();

?>
```

Speichern Sie es unter einem Namen mit der Endung *.php* unterhalb Ihrer `DocumentRoot`, z. B. als *test.php*, und rufen Sie die entsprechende URL mit Ihrem Browser auf: *http://127.0.0.1/test.php*. Wenn alles geklappt hat, sehen Sie nun eine umfangreiche Tabelle, die genaue Auskunft über Ihre PHP- und Apache-Konfiguration gibt.

Installation unter Windows

Die PHP-Installation auf Windows-Rechnern ist ein wenig aufwändiger als unter UNIX, aber dafür geht sie schneller vonstatten, weil Sie das Paket nicht selbst zu kompilieren brauchen. Führen Sie einfach folgende Schritte aus:

1. Entpacken Sie zunächst die ZIP-Datei *php-5.1.2-Win32.zip* in Ihr gewünschtes PHP-Verzeichnis – die folgenden Ausführungen gehen davon aus, dass *C:\php5* verwendet wird.

2. Kopieren Sie die Datei *php.ini-dist* aus dem PHP-Installationsverzeichnis in Ihr Windows-Verzeichnis (je nach Version oft *C:\Windows* oder *C:\WinNT*), und benennen Sie sie dort in *php.ini* um. Öffnen Sie sie mit einem Texteditor, und ändern Sie die beiden folgenden Zeilen:

   ```
   doc_root = "C:\Programme\Apache Group\Apache2\htdocs"
   extension_dir = C:\php5\ext
   ```

   Natürlich müssen Sie Ihre eigene Apache-`DocumentRoot` sowie das korrekte PHP-Verzeichnis angeben.

3. Kopieren Sie die Datei *php5ts.dll* in das Unterverzeichnis *System32* Ihres Windows-Verzeichnisses.

4. Fügen Sie das PHP-Verzeichnis zur Umgebungsvariablen PATH mit den Pfaden für ausführbare Programme hinzu: Wählen Sie *Start* → *Systemsteuerung*, und doppelklicken Sie auf das Symbol *System*. Auf der Registerkarte *Erweitert* finden Sie unten die Schaltfläche *Umgebungsvariablen*. Sie öffnet einen neuen Dialog, in dem Sie unter *Systemvariablen* die Variable *Path* anklicken müssen. Wählen Sie *Bearbeiten*, und fügen Sie an den *Wert der Variablen* ein Semikolon und Ihr PHP-Verzeichnis (zum Beispiel *C:\php5*) an. Nach dreimaligem *OK* zum Schließen der verschachtelten Dialoge müssen Sie Ihren Rechner leider komplett neu starten, damit diese Änderung wirksam wird.

5. Fügen Sie in der Apache-Konfigurationsdatei *httpd.conf* an den passenden Stellen die beiden folgenden Zeilen hinzu (natürlich mit angepassten Pfadangaben):

   ```
   LoadModule php5_module C:/php5/php5apache2.dll
   AddType application/x-httpd-php .php
   ```

Nach einem Neustart von Apache können Sie nun denselben Test durchführen wie unter UNIX.

PHP-Anwendungen und Flash-Filme

Nachdem Sie im vorigen Abschnitt erfahren haben, wie PHP-Anwendungen grundsätzlich funktionieren, geht es hier konkret um die Zusammenarbeit mit Flash-Filmen. Zu Beginn dieses Kapitels wurde bereits angedeutet, dass diese etwas anders funktioniert als die Erstellung HTML-basierter PHP-Anwendungen. Hier noch einmal der schematische Ablauf:

1. Ein Flash-Film dient gewissermaßen als Eingabemaske. Er enthält Textfelder, Komponenten oder selbst erstellte Formularelemente. Er spricht eine PHP-Anwendung an, der er seine Variablen über `POST` oder `GET` sendet. Wenn Sie auch nach dem Versand im aktuellen Flash-Film bleiben möchten, können Sie die Funktion `loadVariablesNum()` benutzen. Wenn Sie eine unabhängige Anwendung aufrufen möchten, wird dagegen `getURL()` verwendet. Dieser »Formular«-Teil einer solchen Anwendung wurde bereits im vorigen Kapitel ausführlich beschrieben.

2. Die PHP-Anwendung nimmt die Variablen des Flash-Films wie HTML-Formulardaten entgegen. Mit diesen Daten führt sie hinter den Kulissen beliebige Operationen durch. Beispiele: Im Falle eines Forums oder Gästebuchs legt sie sie als neuen Beitrag in einer Datenbank ab; bei einer E-Commerce-Anwendung leitet sie sie sogar als Bestellung weiter.
3. Dieser Punkt ist der entscheidende Unterschied zu einer Webanwendung mit HTML-Oberfläche: Die PHP-Anwendung generiert kein vollständiges Dokument, das an den Browser gesendet wird, sondern URL-codierte `Name=Wert`-Paare, die der Flash Player über `loadVariablesNum()` beziehungsweise `loadVariables()` laden kann.
4. Zu guter Letzt wird das »Ergebnis« der Webanwendung angezeigt: Wieder wird `loadVariablesNum()` oder `loadVariables()` verwendet, um die dynamisch erzeugten Daten einer PHP-Anwendung zu laden. Diese Daten können dann beispielsweise zum Ausfüllen dynamischer Textfelder oder für Fallentscheidungen genutzt werden.

Beachten Sie, dass nicht alle Flash-basierten Webanwendungen die hier präsentierte Reihenfolge einhalten müssen. Einige von ihnen benötigen nicht einmal alle diese Aspekte. Beispielsweise könnte sich ein Produktkatalog, der seine Inhalte aus einer Datenbank bezieht, auf den Punkt 4 der Aufzählung beschränken. Umgekehrt bräuchte eine kurzfristige Bestellung – beispielsweise bei einem Pizza-Service – nicht unbedingt eine dynamische Rückmeldung, so dass hier die Punkte 1 und 2 genügen dürften.

Wie PHP funktioniert

Hier soll an einem kurzen Beispiel gezeigt werden, wie PHP-Anwendungen üblicherweise aussehen. Im nächsten Abschnitt geht es nämlich ausschließlich um die Zusammenarbeit mit Flash, die sich von der üblichen dynamischen Erzeugung von HTML-Dokumenten ein wenig unterscheidet.

Das Praktische (manchmal aber auch Unübersichtliche) an PHP ist, dass Sie HTML ganz normal ohne weitere Markierung und ohne Ausgabebefehle benutzen können. Der PHP-Interpreter kümmert sich ausschließlich um Bereiche eines Dokuments, die von <?php und ?> umschlossen werden. Beachten Sie aber, dass einige spezielle PHP-Anweisungen und die zugehörigen <?php...?>-Blöcke in der Datei stehen müssen, bevor die erste HTML-Zeile folgt – selbst ein Leerzeichen vor dem <?php ist schon zu viel. Das liegt daran, dass diese Anweisungen HTTP-Header manipulieren. Bereits im vorigen Kapitel wurde beschrieben, dass der Server diese vor dem eigentlichen Dokument sendet. Beispiele für solche Anweisungen sind `header()` zur direkten Header-Manipulation oder `cookie()` zum Setzen von Cookies.

Die wichtigste PHP-Anweisung ist `echo()` – ihr Argument wird an der aktuellen Position in das Ausgabedokument eingefügt. Eine Besonderheit gegenüber Spra-

→

chen wie ActionScript ist, dass Sie Variablen unmittelbar in die doppelten Anführungszeichen von Strings aufnehmen können: Da in PHP alle Variablennamen mit $ beginnen, weiß der PHP-Interpreter Bescheid. Das folgende Beispiel erzeugt den Ausgabetext *Hallo, Welt!* als HTML-Hauptüberschrift:

```
<?php

   $planet = "Welt";
   echo ("<h1>Hallo, $planet!</h1>");

?>
```

Hier sehen Sie ein kleines Beispiel für ein komplettes PHP-Programm. Es gibt Datum und Uhrzeit aus:

```
<html>
<head>
<title>Datum und Uhrzeit mit PHP</title>
</head>
<body>
<h1>Hallo!</h1>
<?php

   $jetzt = time();
   $datum = date ("d.m.Y", $jetzt);
   $zeit = date ("H:i", $jetzt);
   echo ("Heute ist der $datum. Es ist jetzt genau $zeit Uhr.");
?>
</body>
</html>
```

Die praktische Funktion `date($format, $zeitpunkt)` formatiert Datum und Uhrzeit; die Funktion `time()` dient ihr als Eingabe, weil sie die aktuelle Systemzeit (UNIX-typisch in Sekunden seit EPOCH)[a] ausliest. Eine Liste der zulässigen Formatangaben finden Sie im Verzeichnis *docs* auf der beiliegenden CD-ROM. Abbildung 10-1 zeigt das Ergebnis der Anwendung im Browser.

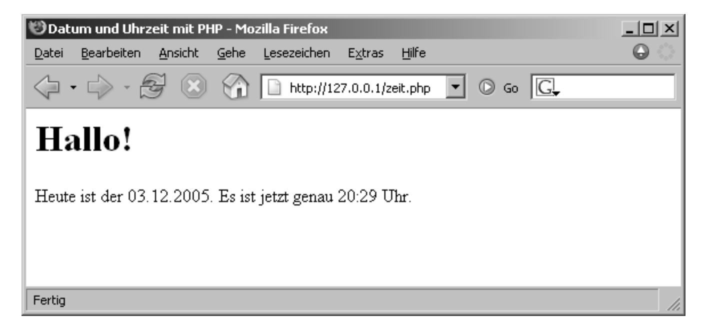


Abbildung 10-1: Ausgabe des PHP-Beispielskripts

→

[a] Idealisiertes UNIX-Erfindungsdatum: 01.01.1970, 00:00 Uhr

Zu guter Letzt sollten Sie noch wissen, wie Formulardaten mit Hilfe von PHP ausgelesen werden, da sie als Eingabedaten für Webanwendungen unentbehrlich sind. Sie stehen in den beiden globalen Arrays `$_POST` und `$_GET` zur Verfügung – je nachdem, mit welcher HTTP-Methode die Formulardaten versendet wurden (siehe voriges Kapitel). Die Indizes dieser Arrays sind die jeweiligen Feldnamen, wie sie über das Attribut `name` der HTML-Formular-Tags oder die Variablennamen eines Flash-Films festgelegt wurden.

Angenommen, ein HTML-Dokument enthält folgendes Formular:

```
<form action="gruss.php" method="get">
  Ihr Name:
  <input type="text" name="user" />
  <input type="submit" value="Absenden" />
</form>
```

Wenn ein Benutzer seinen Namen eintippt und auf *Absenden* klickt, kann das angesprochene PHP-Skript *gruss.php* ihn folgendermaßen persönlich begrüßen:

```
<?php

   $name = $_GET['user'];
   echo ("Hallo, $name!");

?>
```

Flash-Variablen in PHP-Anwendungen laden

Wie bereits erwähnt, können Sie eine PHP-Anwendung aus einem Flash-Film heraus sowohl mit `getURL()` als auch mit einer der `loadVariables*()`-Anweisungen aktivieren. `getURL()` sorgt dafür, dass ein völlig neues Dokument in den Browser geladen wird. Wenn Sie auf diese Weise eine serverseitige Anwendung anfordern, wird erwartet, dass diese Anwendung das Dokument liefert. In Zusammenarbeit mit Flash wäre dies nur dann sinnvoll, wenn der dynamisch erzeugte HTML-Code wieder einen eingebetteten Flash-Film enthielte. Deshalb sind `loadVariablesNum()` beziehungsweise `loadVariables()` oft geeigneter: Sie rufen die Webanwendung zunächst auf und senden ihr die Variablen des aktuellen Kontextes. Daraufhin erwarten sie, dass die Webanwendung ihnen Variablen im URL-Format sendet (siehe Kapitel 8).

Weiter oben wurde bereits angesprochen, wie Sie in einem PHP-Skript auf Formulardaten zugreifen können. Flash-Variablen werden auf dieselbe Weise empfangen. Dabei spielt es keine Rolle, ob sie durch `getURL()` oder `loadVariablesNum()` versandt wurden: Je nach HTTP-Versandmethode liegen die Variablen, die der Flash-Film übermittelt hat, in einem der globalen Arrays `$_GET` beziehungsweise `$_POST` vor. Betrachten Sie als Beispiel folgende ActionScript-Zeilen in einem Flash-Film:

```
var vendor = "Gibson";
var product = "LesPaul";
loadVariablesNum ("catalog.php", 0, "GET");
```

Die `loadVariablesNum()`-Anweisung ruft das PHP-Skript *catalog.php* auf, das sich aufgrund der relativen URL im gleichen Webserver-Verzeichnis befinden muss wie die SWF-Datei selbst. In der PHP-Anwendung können Sie die beiden Variablen `vendor` und `product` folgendermaßen aus `$_GET` auslesen:

```
$vendor = $_GET['vendor'];
$product = $_GET['product'];
```

Natürlich brauchen Sie in PHP nicht unbedingt identische Variablennamen – hier `$vendor` und `$product` – zu wählen. Die entsprechenden Variablen sollten übrigens auch dann definiert sein, wenn die SWF-Datei sie wider Erwarten nicht liefert – beispielsweise, wenn Sie sich beim Variablennamen verschrieben haben oder wenn ein Benutzer ein Textfeld nicht ausgefüllt hat. Zu diesem Zweck können Sie folgende Ausdrucksweise verwenden:

```
$vendor = "";
if (isset ($_GET['vendor']) && $_GET['vendor'] != "") {
   $vendor = $_GET['vendor'];
}
```

Diese Schreibweise hat den Vorteil, dass sie mit unterschiedlichen PHP-Versionen gleichermaßen funktioniert. In älteren PHP-Versionen sind undefinierte Variablen nämlich leer – `$variable == ""` ergibt `true`. Neuere Versionen reagieren dagegen auf `isset()`. Falls Ihr Hoster also eine ältere PHP4-Version benutzen sollte, brauchen Sie Ihre Skripte dafür nicht umzuschreiben.

Wenn Sie viele Variablen auslesen müssen, kann es ziemlich lästig sein, diese Zeilen für jede einzelne Variable zu schreiben. Deshalb lohnt es sich, für diese Aufgabe folgende Funktion zu definieren:

```
function cgiParam ($field, $default)
{
   // Variable auf Standardwert setzen
   $var = $default;
   // HTTP-Methode ermitteln
   $meth = $_SERVER['REQUEST_METHOD'];
   if ($meth == "GET") {
      if (isset ($_GET[$field]) && $_GET[$field] != "") {
         $var = $_GET[$field];
      }
   } elseif ($meth == "POST") {
      if (isset ($_POST[$field]) && $_POST[$field] != "") {
         $var = $_POST[$field];
      }
   }
   // Variable zurückgeben
   return $var;
}
```

Wie Sie erkennen können, sehen Funktionen in PHP exakt genauso aus wie in ActionScript. Die Funktion cgiParam() nimmt zwei Werte entgegen: den Namen des Formularfeldes beziehungsweise der Flash-Variablen, die Sie auslesen möchten ($field), sowie den Standardwert $default, der zurückgegeben wird, wenn das Feld nicht vorhanden oder leer ist. Zunächst wird aus dem Array $_SERVER die Umgebungsvariable REQUEST_METHOD ausgelesen; sie enthält je nach Zugriffsmethode einen der Werte "GET" oder "POST". Anschließend erfolgt der Versuch, die gewünschte Variable aus dem passenden Array auszulesen. Zu guter Letzt liefert return() den gefundenen Wert zurück.

Um nun die Variablen vendor und product mit Hilfe dieser Funktion zu lesen, können Sie die beiden folgenden Zeilen verwenden:

```
$vendor = cgiParam ("vendor", "");
$product = cgiParam ("product", "");
```

Damit Ihnen diese nützliche Funktion jederzeit zur Verfügung steht, sollten Sie sie in einer externen PHP-Include-Datei abspeichern. Solche Dateien haben für gewöhnlich die Dateiendung .inc.php; auch darin müssen Sie PHP-Anweisungen in einen <?php...?>-Block setzen. Sie können eine solche Datei wie folgt einbetten:

```
include ("mylib.inc.php");
```

PHP-Daten in Flash-Filme laden

Während Sie nun bereits wissen, wie Ihre PHP-Anwendung Flash-Daten entgegennehmen kann, ist der umgekehrte Weg noch unbekannt. Die zuständigen loadVariables*()-Funktionen haben Sie zwar bereits kennen gelernt, in Kapitel 8 wurde aber lediglich ihre Verwendung mit statischen Textdateien demonstriert. Damit Sie auch dynamische Daten damit laden können, muss ein entsprechendes Skript das passende Textformat als Ausgabe erzeugen. Dazu müssen Sie zunächst den MIME-Type text/plain setzen. In PHP geschieht dies mit Hilfe der Anweisung header() – wie bereits erwähnt, darf vor dem anfänglichen PHP-Block, in dem dies geschieht, kein einziges Zeichen stehen. Fügen Sie einfach folgende Zeile in Ihr Skript ein:

```
header ("Content-type: text/plain");
```

Alternativ können Sie übrigens auch den von HTML-Formularen bekannten Typ application/x-www-form-urlencoded benutzen. Anschließend können Sie den Ausgabebefehl echo() oder einfachen Text außerhalb von PHP-Blöcken verwenden, um eine Abfolge von Variablen in der Form var1=Wert1&var2=Wert2&... auszugeben. Die Werte der Variablen sollten dabei, genau wie bei Formularfeldern, URL-codiert sein. Zu diesem Zweck definiert PHP die Funktion urlencode($wert). Das folgende Komplettbeispiel übermittelt das aktuelle Datum in der Form 03.12.2005 und die Uhrzeit im Format 19:34:

```
<?php

   // MIME-Type text/plain setzen
   header ("Content-type: text/plain");

   // Datum und Uhrzeit formatieren
   $jetzt = time();
   $datum = urlencode (date ("d.m.Y", $jetzt));
   $zeit = urlencode (date ("H:i", $jetzt));

   // Ausgabe
   echo ("datum=$datum&zeit=$zeit");

?>
```

Speichern Sie das Skript – zum Beispiel unter dem Namen *zeit.php* – im Website-Verzeichnis Ihres Webservers. Zur Kontrolle können Sie es in einem Browser aufrufen und erhalten eine Ausgabe wie diese:

```
datum=03.12.2005&zeit=19%3A34
```

Als Nächstes benötigen Sie einen Flash-Film, der auf das PHP-Skript zugreift und diese Variablen lädt. Am praktischsten ist es, wenn Sie zwei dynamische Textfelder erstellen und ihnen die Variablennamen `datum` beziehungsweise `zeit` zuordnen – dann brauchen Sie außer der `loadVariablesNum()`-Anweisung keinen weiteren Code mehr, um das Ergebnis anzuzeigen. Wenn Sie die SWF-Datei in dasselbe Verzeichnis Ihrer Site veröffentlichen wie das Skript, genügt dazu folgende Zeile:

```
loadVariablesNum ("zeit.php", 0, "GET");
```

Sollte das Skript dagegen in einem anderen Verzeichnis liegen oder gar auf einem anderen Server ausgeführt werden, dann müssen Sie seine URL entsprechend anpassen. Das zweite Argument ist die von `loadMovieNum()` bekannte Stufe – wenn nur ein einzelner Hauptfilm beteiligt ist, bleibt es bei der hier verwendeten 0. Die Methode schließlich spielt für das Laden der Variablen selbst keine Rolle. Der Server sendet sowohl auf `POST`- als auch auf `GET`-Anfragen eine gewöhnliche HTTP-Antwort, bei der der Header `Content-Type` den Datentyp angibt, während die eigentlichen Daten im Body enthalten sind. Die Angabe von `"POST"` oder `"GET"` bestimmt also nur, auf welche Weise die Variablen Ihres Flash-Films an die Server-Anwendung gesendet werden sollen.

Mehr Kontrolle: Die Klasse LoadVars

Es gibt noch eine leistungsfähigere Alternative zum Einsatz von `loadVariablesNum()`: Sie können eine Instanz der Klasse `LoadVars` erstellen, mit der sich das Laden sowie das Senden von Variablen genauer steuern lässt. Der Konstruktor selbst hat keine Argumente; das Objekt wird stets auf folgende Weise erstellt:

```
var loader = new LoadVars();
```

Die einzelnen Lade- und Sendefunktionen werden durch die Methoden dieser Instanz bereitgestellt. Die wichtigsten von ihnen sind folgende:

- `loader.load(URL)` lädt Variablen aus der angegebenen URL.
- `loader.send(URL, [Ziel, Methode])` sendet die Variablen des aktuellen Kontextes an die angegebene URL. Zusätzlich können Sie als *Ziel* ein Zielfenster oder -frame angeben, in dem dann die Server-Antwort angezeigt wird – in diesem Fall funktioniert die Methode wie `getURL()`. Falls Sie ein Ziel definieren, steht es Ihnen frei, als drittes Argument die gewünschte HTTP-Versandmethode anzugeben; Standard ist ansonsten `POST`.
- `loader.sendAndLoad(URL, Zielobjekt[, Methode])` sendet zunächst die Variablen an die angegebene URL. Anschließend nimmt `sendAndLoad()` die Antwort des Servers entgegen und speichert sie als Variablen in dem `LoadVars`-Objekt *Zielobjekt*. Im Grunde entspricht diese Methode also am ehesten dem Standardverhalten von `loadVariablesNum()`.

Diese drei Methoden bieten noch nichts, was sich nicht mit `loadVariablesNum()` oder `getURL()` machen ließe. Der eigentliche Vorteil von `LoadVars`-Instanzen besteht darin, dass Sie eine Kontrolle über den Ladevorgang selbst ausüben können. Zu diesem Zweck stehen Ihnen einige zusätzliche Methoden und Eigenschaften zur Verfügung. Beispielsweise können Sie über die Methode `contentType()` bestimmen, welchen Datentyp die mittels `POST` gesendeten Variablen haben sollen. Der Standardtyp ist wie bei einem HTML-Formular die einfache URL-Codierung (`application/x-www-form-urlencoded`). Für spezielle Anwendungen kann aber ein anderer MIME-Type nützlicher sein. Das folgende Beispiel legt einfachen Text (`text/plain`) als Format fest:

```
loader.contentType ("text/plain");
```

Ein weiteres Hilfsmittel ist die Eigenschaft `loaded`: Sie erhält den Wert `true`, sobald die Variablen fertig geladen sind. Das folgende Beispiel bewirkt einen Sprung zu einem Frame namens `"showData"`, sobald die Instanz `loader` ihre Arbeit beendet hat:

```
if (loader.loaded) {
   gotoAndPlay ("showData");
}
```

Auf diese Weise lassen sich auch für externe Daten Preloader-Schleifen (siehe Kapitel 8) erstellen. Wenn Sie gar eine Prozentanzeige oder einen Fortschrittsbalken verwenden möchten, können Sie die beiden Methoden `getBytesLoaded()` und `getBytesTotal()` zueinander in Beziehung setzen, die die Anzahl der bereits geladenen beziehungsweise der insgesamt erwarteten Bytes liefern. Das folgende Beispiel gibt den entsprechenden Prozentwert in ein Textfeld aus, das mit einer Variablen namens `loadPercent` verknüpft ist:

```
loadPercent =
   parseInt (100 / loader.getBytesTotal() * getBytesLoaded());
```

Zu guter Letzt können Sie noch den nützlichen Event-Handler `onLoad` verwenden, indem Sie ihm eine Funktion zuordnen. Dies erspart Ihnen eine Schleife mit Sprungbefehlen, weil die Funktion genau dann aktiviert wird, wenn ein `load()`-beziehungsweise `sendAndLoad()`-Vorgang abgeschlossen ist. Dabei wird dem Handler der Wert `true` übergeben, wenn das Laden Erfolg hatte, oder `false`, wenn es mit einem Fehler abgebrochen wurde. Hier sehen Sie ein einfaches Beispiel:

```
// LoadVars-Instanz definieren
var loader = new LoadVars();

// Ladevorgang einleiten
loader.load ("lotsofvars.php");

// Event-Handler deklarieren - wird nach dem Laden aufgerufen
loader.onLoad = function(success) {
   if (success) {
      // Erfolgreich geladen
      gotoAndPlay ("showData");
   } else {
      // Ladefehler
      gotoAndPlay ("fehler");
   }
}
```

ActionScript und XML

Die *Extensible Markup Language* (XML) hat sich in den letzten Jahren zum wichtigsten Datenaustauschformat für verteilte Anwendungen entwickelt. Flash kann XML-Dateien gemäß dem DOM-Standard (Document Object Model) des WWW-Konsortiums verarbeiten. Auf diese Weise können Sie XML-Dokumente laden und in Form eines Baummodells auswerten, aber auch modifizieren und neu konstruieren. Der kurze Einstieg in diesem Abschnitt beschränkt sich auf die Verarbeitung vorhandener XML-Dokumente.



Auf der Basis des XML-Datenformats wurden die *Web Services* entwickelt – die automatisierte Zusammenarbeit von Anwendungen per HTTP-Kommunikation, wobei Plattform, Betriebssystem und Programmiersprache so gut wie keine Rolle spielen. Auf der Website zum Buch finden Sie ein Beispiel, das die Zusammenarbeit einer Flash-Anwendung mit dem öffentlich verfügbaren Google Web Service demonstriert.

XML-Dokumente

XML ist zunächst keine Sprache mit speziellen Schlüsselwörtern, die eine konkrete Bedeutung haben, sondern ein Metaformat zur Definition solcher Sprachen. XHTML ist beispielsweise die XML-basierte Neufassung von HTML. Jenseits sol-

cher Standards können Sie Dokumente mit beliebigen Tags schreiben; solange Sie sich dabei an bestimmte Regeln halten, handelt es sich um *wohlgeformte XML-Dokumente*. Diese Regeln sind:

- Als Einleitung wird eine Steueranweisung mit dem folgenden Schema benötigt:
  <?xml version="1.0" encoding="utf-8" ?>
  Statt utf-8 sollte der tatsächliche Zeichensatz des Dokuments angegeben werden, zum Beispiel iso-8859-1.
- Abgesehen von der Einleitungszeile muss ein Dokument vollständig von einem einzelnen Tag umschlossen werden, dem so genannten *Wurzelelement*. Bei XHTML-Dokumenten ist dies beispielsweise <html>...</html>.
- Jedes Element besteht aus einem Start-Tag und einem End-Tag. Das Start-Tag hat die Form <Tagname [Attribut="Wert" ...]>; das End-Tag lautet immer einfach </Tagname>, ohne Wiederholung der Attribute. Wenn ein Element keine Inhalte umschließt, ist die Kurzschreibweise <Tagname [Attribut="Wert" ...] /> für die Kombination aus öffnendem und schließendem Tag zulässig – ein Beispiel wäre das XHTML-Tag
 für einen einfachen Zeilenumbruch.
- In Elemente können beliebig viele weitere Elemente oder einfacher Text (PCDATA) verschachtelt werden.
- Zeichen, die in der XML-Syntax eine spezielle Bedeutung haben, müssen durch so genannte *Entity-Referenzen* ersetzt werden: < steht für <, > für >, & für &, " für " und ' für '.

Um diese Regeln transparent zu machen, sehen Sie hier einen Ausschnitt aus einer XML-Repräsentation der Struktur dieses Buches (die gekürzten Stellen werden durch <!-- Kommentare --> gekennzeichnet):

```
<?xml version="1.0" encoding="utf-8" ?>
<buch isbn="3-89721-277-3">
  <titel>Praxiswissen Flash 8</titel>
  <autor>Sascha Kersken</autor>
  <verlag>O&apos;Reilly</verlag>
  <kapitel>
    <titel>Flash im Überblick</titel>
    <abschnitt>Was ist Flash?</abschnitt>
    <abschnitt>Flash installieren</abschnitt>
    <abschnitt>Die Arbeitsumgebung von Flash</abschnitt>
    <abschnitt>Praxisbeispiel: Der erste Flash-Film</abschnitt>
  </kapitel>
  <kapitel>
    <titel>Zeichnen mit Flash</titel>
    <!-- Liste der Abschnitte in Kap. 2 -->
  </kapitel>
  <!-- Hier weitere Kapitel -->
</buch>
```

XML verarbeiten

Als Beispiel für die Verarbeitung von XML-Daten wird hier aus einem XML-Dokument ein verschachteltes Hyperlink-Menü erzeugt. Nebenbei erfahren Sie, wie eine Movieclip-Instanz dynamisch dupliziert werden kann.

Öffnen Sie die Datei */beispiele/kapitel10/xml_menu.fla* von der beiliegenden CD-ROM. Die Anwendung ist bereits fertig, so dass es hier keine Praxisanleitung, sondern lediglich eine Beschreibung gibt. Kopieren Sie die Datei *menu.xml* aus demselben Verzeichnis in das Wurzelverzeichnis Ihres Webservers (zum Beispiel *C:\Programme\Apache Group\Apache2\htdocs*; siehe oben). Erst dann können Sie den Film testen.

Auf der Bühne befindet sich eine einzelne Movieclip-Instanz mit dem Namen `item`. Wenn Sie ihre Eigenschaften betrachten, stellen Sie fest, dass es sich um eine Instanz von `menuitem` handelt. Rufen Sie daher zunächst die Definition des Symbols `menuitem` auf. Hier finden Sie die vier Ebenen *hg* (Hintergrundfarbverlauf), *txt* (enthält ein dynamisches Textfeld mit dem Instanznamen `txtbox` und der Variablenzuordnung `cap` für die dynamische Beschriftung), *btn* (Instanz des transparenten Schaltflächensymbols *button*) und *action* (Bildaktion).

Die Bildaktion erstellt lediglich eine Color-Instanz zum dynamischen Einfärben der Beschriftung bei Mausberührung:

```
var c = new Color (this.txtbox);
```

Etwas umfangreicher ist das Skript, das der Schaltfläche zugeordnet wurde:

```
on (rollOver) {
   c.setRGB (0xFF0000);
}
on (rollOut) {
   c.setRGB (0x000000);
}
on (release) {
   getURL (this.href);
}
```

Sobald die Maus die Schaltfläche berührt, wird der Text rot gefärbt; beim Verlassen wird er wieder schwarz. Klicken und Loslassen ruft die in der Variablen `href` enthaltene URL auf – der Wert dieser Variablen wird für die einzelnen Instanzen aus dem XML-Dokument ausgelesen.

Der gesamte Rest der Anwendung findet im Skript des ersten Bildes im Hauptfilm statt. Hier wird zunächst eine Funktion namens `getMenuItems()` definiert, deren Beschreibung weiter unten folgt. Zuallererst wird nämlich die ursprüngliche `menuitem`-Instanz unsichtbar geschaltet:

```
_root.item._visible = false;
```

Anschließend muss menu.xml geladen und hierarchisch durchwandert werden. Dazu wird eine Instanz der Klasse XML erstellt; ihre Methode load() lädt eine XML-URL:

```
var menu:XML = new XML();
menu.load("http://localhost/menu.xml");
```

Damit die Verarbeitung erst nach dem vollständigen Laden beginnt, erfolgt der Aufruf von getMenuItems() innerhalb eines onLoad-Event-Handlers, und zwar nur im Erfolgsfall:

```
menu.onLoad = function(success) {
   if (success) {
      getMenuItems(menu, 0, 0);
   } else {
      trace("Sorry! Kein Menü!");
   }
};
```

Bevor die Vearbeitung von menu.xml durch die Funktion getMenuItems() beschrieben wird, sollten Sie sich diese Datei selbst anschauen. Sie besteht aus dem Wurzelelement <menu> sowie einer Reihe beliebig tief ineinander verschachtelter <item>-Tags, in denen das Attribut href für die jeweilige Link-Adresse und caption für die Beschriftung steht. Mit der Website von O'Reilly sowie meiner eigenen als Beispielen sieht diese Datei so aus:

```
<?xml version="1.0" encoding="utf-8" ?>
<menu>
  <item href="http://www.oreilly.de" caption="O&apos;Reilly">
    <item href="http://www.oreilly.de/basics"
     caption="Reihe Basics" />
    <item href="http://www.oreilly.de/web" caption="Thema Web" />
  </item>
  <item href="http://buecher.lingoworld.de" caption="Sascha Kersken">
    <item href="http://buecher.lingoworld.de/flash"
     caption="Flash" />
    <item href="http://buecher.lingoworld.de/apache2"
     caption="Apache 2" />
    <item href="http://buecher.lingoworld.de/mysql"
     caption="MySQL" />
  </item
</menu>
```

Die Funktion getMenuItems(), die diese Datei auswertet und die entsprechenden Menüelemente zeichnet, arbeitet *rekursiv* – das bedeutet, dass sie sich für die untergeordneten XML-Elemente jeweils selbst aufruft. Dies stellt sicher, dass die Hierarchie korrekt abgearbeitet wird. Die Funktion nimmt drei Parameter entgegen: den DOM-XML-Knoten (Klasse XMLNode) x, der einen Bestandteil des XML-Dokuments wie ein Element oder einfachen Text repräsentiert, die numerische Einrückungstiefe indent sowie die aktuelle Zeile row, ebenfalls eine Zahl:

```
function getMenuItems(x:XMLNode, indent:Number, row:Number):Number {
```

Der Rückgabetyp der Funktion ist Number, nämlich die jeweils nächste Zeilennummer. Innerhalb des Funktionrumpfes wird zunächst überprüft, ob der aktuelle Knoten x ein Menüpunkt ist. Dazu wird seine Eigenschaft nodeType getestet – 1 ist ein XML-Element, während 3 für normalen Text steht. Außerdem muss die Eigenschaft nodeName, in diesem Fall der Elementname, den Wert "item" haben:

```
if (x.nodeType == 1 && x.nodeName == "item") {
```

Ist beides der Fall, dann werden zunächst die Werte der beiden Attribute href und caption ermittelt – dazu dient das Array attributes, dessen Indizes die Attributnamen sind:

```
var href:String = x.attributes["href"];
var cap:String = x.attributes["caption"];
```

Die nächste Zeile erstellt eine Kopie der Movieclip-Instanz item. Die Methode *clip*.duplicateMovieClip() benötigt als Argumente einen individuellen Namen sowie eine Tiefe (Darstellungsstufe, die die Überlagerung auf der Bühne regelt) – beides wird hier mit Hilfe der Variablen row gebildet, so dass die einzelnen Clip-Kopien item0, item1 und so weiter heißen und entsprechend durchnummerierte Tiefen besitzen. Die Kopieranweisung lautet insgesamt:

```
var theItem:MovieClip =
     _root.item.duplicateMovieClip("item" + row, row);
```



Sie müssen streng darauf achten, jede Tiefe nur einmal zu verwenden – andernfalls werden vorhandene Movieclip-Instanzen ersetzt.

Die Variable theItem verweist auf die neu erstellte Kopie, damit im Folgenden ihre Eigenschaften und Variablen modifiziert werden können. Als Erstes wird ihre x-Position berechnet; es handelt sich um die Summe aus 10 Pixeln vom linken Rand entfernt und dem Dreißigfachen der aktuellen Einrückungsstufe:

```
theItem._x = 30 * indent + 10;
```

Die y-Position hat ebenfalls einen Abstand von zehn Pixeln vom oberen Rand; pro Zeile werden die Höhe der Instanz und 10 Pixel Abstand hinzuaddiert:

```
theItem._y = (theItem._height + 10) * row + 10;
```

Als Nächstes werden die beiden Clip-Variablen für die Verknüpfungs-URL beziehungsweise die Beschriftung gesetzt:

```
theItem.href = href;
theItem.cap = cap;
```

Nun wird die Instanz, die nun das richtige Aussehen und die korrekte Position besitzt, sichtbar gemacht:

```
theItem._visible = true;
```

Zu guter Letzt wird die Zeilennummer um 1 erhöht, da soeben erfolgreich ein Menüpunkt angezeigt wurde:

```
row++;
```

Der Rest des Codes muss für jeden Knoten ausgeführt werden, also auch dann, wenn es kein Menüpunkt war. Mit Hilfe der `XMLNode`-Eigenschaft `hasChildNodes` wird überprüft, ob es untergeordnete XML-Knoten gibt:

```
if (x.hasChildNodes) {
```

In diesem Fall werden diese in einer Schleife durchwandert:

```
for (var j:Number = 0; j < x.childNodes.length; j++) {
```

Für jeden Kindknoten erfolgt ein verschachtelter Aufruf von `getMenuItems()`. Als Parameter werden der aktuelle Kindknoten, die um 1 erhöhte Einrückung sowie die aktuelle Zeile übergeben. Das Ergebnis wird wieder in `row` gespeichert, um die Zeilennummer aktuell zu halten:

```
row = getMenuItems(x.childNodes[j], indent + 1, row);
```

Die letzte Zeile der Funktion ist das Gegenstück zu diesem Aufruf – sie gibt den aktuellen Wert von `row` zurück:

```
return row;
```

Hier noch einmal der komplette, kommentierte Code des Bildskriptes:

```
/* Aktuellen XML-Knoten überprüfen. Falls es ein
   Menüelement ist, einen entsprechenden Movieclip
   erzeugen. Anschließend die Kindknoten abarbeiten. */
function getMenuItems(x:XMLNode, indent:Number, row:Number):Number {
   // Ist der aktuelle Knoten ein <item>?
   if (x.nodeType == 1 && x.nodeName == "item") {
      // Attribute Link-URL und Beschriftung auslesen:
      var href:String = x.attributes["href"];
      var cap:String = x.attributes["caption"];
      // Movieclip-Kopie erstellen
      var theItem:MovieClip =
           _root.item.duplicateMovieClip("item" + row, row);
      // x- und y-Position nach Einrückstufe bzw. Zeile berechnen:
      theItem._x = 30 * indent + 10;
      theItem._y = (theItem._height + 10) * row + 10;
      // URL und Beschriftung des Clips setzen:
      theItem.href = href;
      theItem.cap = cap;
      // Die fertige Clip-Kopie sichtbar machen:
      theItem._visible = true;
      // Zeilennummer erhöhen, da ein Menüpunkt erstellt wurde:
      row++;
   }
```

```
      // Gibt es Kindknoten?
      if (x.hasChildNodes) {
         // Schleife über alle Kindknoten:
         for (var j:Number = 0; j < x.childNodes.length; j++) {
            // Rekursiver Aufruf für den aktuellen Kindknoten:
            row = getMenuItems(x.childNodes[j], indent + 1, row);
         }
      }
      // Aktuelle Zeilennummer zurückgeben:
      return row;
   }
   // Den Original-Movieclip unsichtbar machen:
   _root.item._visible = false;
   // Neues XML-Objekt erzeugen:
   var menu:XML = new XML();
   // XML-Datei laden:
   menu.load("http://localhost/menu.xml");
   // Sobald die Datei fertig geladen ist:
   menu.onLoad = function(success) {
      if (success) {
         // Im Erfolgsfall den XML-Baum durcharbeiten:
         getMenuItems(menu, 0, 0);
      } else {
         // Andernfalls Fehlermeldung:
         trace("Sorry! Kein Menü!");
      }
   };
```

Die Ausgabe des Films können Sie in Abbildung 10-2 betrachten.



Abbildung 10-2: Das XML-Menü im Einsatz

Praxisbeispiel: Bestellannahme und Gästebuch

Wie Sie bemerkt haben dürften, enthält dieses Kapitel viele Beschreibungen allgemeiner Konzepte und Vorbereitungen, aber nur wenig an theoretischen Grundlagen zur Zusammenarbeit zwischen PHP und Flash selbst. Das liegt daran, dass die Schnittstelle selbst im Grunde sehr einfach ist und immer auf dieselbe Weise funktioniert. Umso wichtiger ist es deshalb, dass Sie konkrete Beispiele für ihren Einsatz kennen lernen. Daher erhält die Website des FlashRock Music Shop nun den letzten Schliff: Es wird eine PHP-Anwendung vorgestellt, die das Bestellformular aus dem vorigen Kapitel auswertet. Zum Schluss wird noch ein datenbankbasiertes Gästebuch hinzugefügt.

Verarbeitung der Online-Bestellungen

Im Praxisteil des vorigen Kapitels wurde unter anderem ein einfaches Flash-Formular für Online-Bestellungen gezeigt. Hier wird ein PHP-Skript vorgestellt, das die Bestelldaten entgegennimmt. Es addiert alle Preise zusammen, weist den korrekten Mehrwertsteueranteil aus (wobei der Einfachheit halber alle Artikel mit 16% Mehrwertsteuer belegt werden), entscheidet über Versandkosten (ab 50,- Euro frei) und gibt dieses Ergebnis als einfaches HTML-Dokument aus. Damit dieses Skript aktiviert werden kann, müssen Sie den Film (*beispiele/kapitel10/form_ziel.fla* auf der beiliegenden CD-ROM) um ein kleines Skript erweitern. Weisen Sie dem (bisher nur als Dummy verfügbaren) Button *Abschicken* folgendes Skript zu:

```
on (release) {
   getURL ("order.php", "_self", "GET");
}
```

Das Skript *order.php* muss in demselben Verzeichnis Ihres Webservers gespeichert werden wie die veröffentlichte SWF-Datei. Es besitzt folgenden Inhalt, den Sie in einem beliebigen Texteditor eingeben können:

```
<?php

   // Funktion zum Lesen von Parametern

   function cgiParam ($field, $default)
   {
      // Variable auf Standardwert setzen
      $var = $default;
      // HTTP-Methode ermitteln
      $meth = $_SERVER['REQUEST_METHOD'];
      if ($meth == "GET") {
         if (isset ($_GET[$field]) && $_GET[$field] != "") {
            $var = $_GET[$field];
         }
```

```
      } elseif ($meth == "POST") {
         if (isset ($_POST[$field]) && $_POST[$field] != "") {
            $var = $_POST[$field];
         }
      }
      // Variable zurückgeben
      return $var;
   }

   // Daten einlesen: Artikel-Stückzahlen
   $stratStk = cgiParam ("strat", 0);
   $yamaStk = cgiParam ("yama", 0);
   $ampStk = cgiParam ("amp", 0);
   $kabStk = cgiParam ("kab", 0);
   $buchStk = cgiParam ("buch", 0);

   // Daten einlesen: Persönliche Daten / Bestellweg
   $kunde = cgiParam ("kunde", "");
   $adr = cgiParam ("adr", "");
   $plz = cgiParam ("plz", "");
   $ort = cgiParam ("ort", "");
   $pay = cgiParam ("pay", "rech");
   $nr = cgiParam ("nr", "");

?>
<html>
  <head>
    <title>FlashRock Music Shop - Bestellbest&auml;tigung</title>
  </head>
  <body>
    <h1>Vielen Dank f&uuml;r Ihre Bestellung</h1>
    <table border="2">
      <tr>
        <th>St&uuml;ck</th>
        <th>Artikel</th>
        <th>Einzelpreis</th>
        <th>Gesamtpreis</th>
      </tr>
<?php

   $summe = 0;

   if ($stratStk) {
      // Stratocaster gekauft
      $stratPr = $stratStk * 599;
      $summe += $stratPr;
      echo ("<tr>\n");
      echo ("<td>$stratStk</td>\n");
      echo ("<td>Fender Stratocaster E-Gitarre</td>\n");
      echo ("<td align=\"right\">599,-- EUR</td>\n");
      echo ("<td align=\"right\">$stratPr,-- EUR</td>\n");
      echo ("</tr>\n");
   }
```

```
if ($yamaStk) {
   // Yamaha gekauft
   $yamaPr = $yamaStk * 399;
   $summe += $yamaPr;
   echo ("<tr>\n");
   echo ("<td>$yamaStk</td>\n");
   echo ("<td>Yamaha Halbakustik-Gitarre</td>\n");
   echo ("<td align=\"right\">399,-- EUR</td>\n");
   echo ("<td align=\"right\">$yamaPr,-- EUR</td>\n");
   echo ("</tr>\n");
}
if ($ampStk) {
   // Verstärker gekauft
   $ampPr = $ampStk * 599;
   $summe += $ampPr;
   echo ("<tr>\n");
   echo ("<td>$ampStk</td>\n");
   echo ("<td>Marshall Kofferverst&auml;rker</td>\n");
   echo ("<td align=\"right\">299,-- EUR</td>\n");
   echo ("<td align=\"right\">$ampPr,-- EUR</td>\n");
   echo ("</tr>\n");
}
if ($kabStk) {
   // Kabel gekauft
   $kabPr = $kabStk * 14;
   $summe += $kabPr;
   echo ("<tr>\n");
   echo ("<td>$kabStk</td>\n");
   echo ("<td>Gitarrenkabel 3 m</td>\n");
   echo ("<td align=\"right\">14,-- EUR</td>\n");
   echo ("<td align=\"right\">$kabPr,-- EUR</td>\n");
   echo ("</tr>\n");
}
if ($buchStk) {
   // Lehrbuch gekauft
   $buchPr = $buchStk * 19;
   $summe += $buchPr;
   echo ("<tr>\n");
   echo ("<td>$buchStk</td>\n");
   echo ("<td>Buch &quot;Rock-Gitarre&quot;</td>\n");
   echo ("<td align=\"right\">19,-- EUR</td>\n");
   echo ("<td align=\"right\">$buchPr,-- EUR</td>\n");
   echo ("</tr>\n");
}

// Versandkostenpflichtig?
if ($summe < 50) {
   $summe += 5;
   echo ("<tr><td colspan=\"3\">zzgl. Versandkosten</td>");
   echo ("<td align=\"right\">5,-- EUR</td></tr>\n");
}
```

```
    // Gesamtsumme
    echo ("<tr><td colspan=\"3\"><b>Gesamtsumme:</b></td>");
    echo ("<td align=\"right\"><b>$summe,-- EUR</b></td></tr>\n");

    // Mwst.
    $mwst = round ($summe / 116 * 16, 2);
    echo ("<tr><td colspan=\"3\">incl. 16% Mehrwertsteuer</td>");
    echo ("<td align=\"right\">$mwst EUR</td></tr>\n");
?>
    </table>
    <br />
    Lieferung frei Haus an:<br /><br />
<?php

    echo ("$kunde<br />");
    echo ("$adr<br />");
    echo ("$plz $ort<br /><br />");
    if ($pay == "rech") {
       echo
       ("Die Lieferung erfolgt incl. Rechnung &uuml;ber $summe,--.");
    } else {
       echo ("Den Betrag in H&ouml;he von <b>$summe,-- EUR</b>
 buchen wir von Ihrem Konto Nr. $nr ab.");
    }
?>
  </body>
</html>
```

Abbildung 10-3 zeigt die Ausgabe des Skripts nach der Ausführung einer Bestellung.

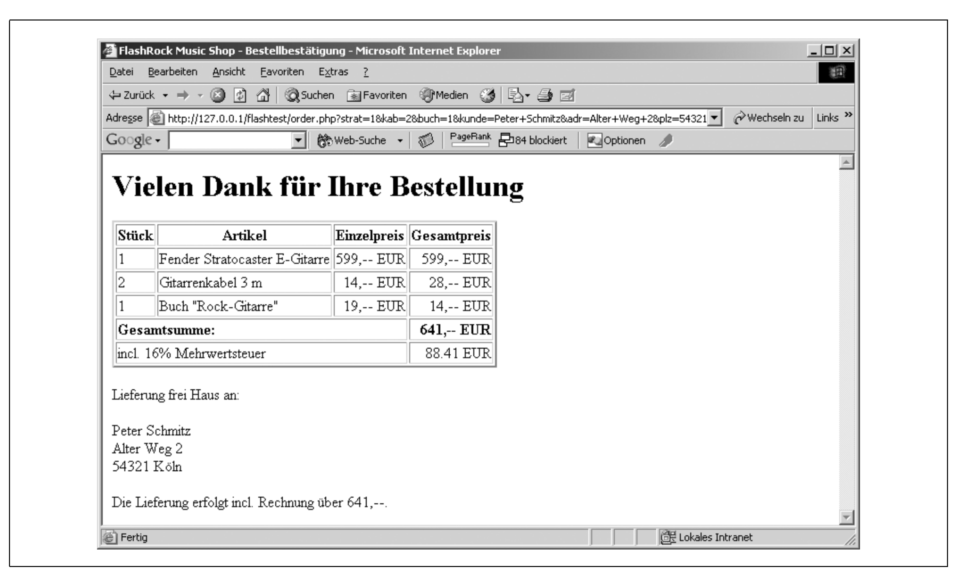


Abbildung 10-3: Bestellbestätigung durch ein PHP-Skript

Ein Gästebuch

Viele private, aber auch einige kommerzielle Websites sind seit Jahren mit virtuellen Gästebüchern ausgestattet. Passend zum Rest seiner Website erhält der FlashRock Music Shop natürlich ein Gästebuch, dessen Oberfläche vollständig in Flash erstellt wurde. Die Funktionalität hinter den Kulissen stellen zwei kleine PHP-Skripte zur Verfügung: Das eine liefert auf Anforderung einen bestimmten nummerierten Eintrag aus dem Gästebuch, während das andere einen neuen Eintrag im Gästebuch speichert. Die Daten werden in einer kleinen MySQL-Datenbanktabelle gespeichert.

Die fertige Benutzeroberfläche des Gästebuches finden Sie auf der beiliegenden CD-ROM. Sie befindet sich im Verzeichnis *beispiele/kapitel10* und heißt *guest_start.fla*. Wenn Sie Ihre fertige Datei überprüfen möchten, können Sie die vorgefertigte Version *guest_ziel.fla* laden.

PHP-Zugriff auf MySQL-Datenbanken

Weiter oben wurde bereits kurz erläutert, wie PHP arbeitet und wie Sie MySQL-Datenbanken erstellen können. Hier soll nun angesprochen werden, wie Sie aus Ihren PHP-Skripten heraus auf MySQL-Daten zugreifen können. Ein kleines Problem besteht darin, dass es zwei verschiedene Schnittstellen dafür gibt: Wenn Sie PHP ab 4.3.x und MySQL ab 4.1 einsetzen, können Sie die neue `mysqli`-Schnittstelle benutzen, andernfalls (insbesondere bei MySQL 4.0.x) sind Sie auf den klassischen Zugriff angewiesen. Hier wird nur die neue Methode erläutert. Beachten Sie aber bitte, dass die Versionen bei manchen Webhostern noch nicht neu genug sind; für diesen Fall finden Sie auf der Website zum Buch eine Anleitung zur Verwendung der alten Schnittstelle.

Zuerst müssen Sie eine Verbindung zum Datenbank-Server herstellen. Dazu benötigen Sie dessen Domain-Namen oder IP-Adresse; daneben sind Benutzername und Passwort erforderlich. Wenn Sie sich an die Installationsanleitung weiter oben gehalten haben, laufen Datenbank- und Webserver auf demselben Rechner. Angenommen, Sie verwenden den oben erstellten Benutzer *dbuser* mit dem Passwort *geheim* und möchten auf die Datenbank `musik` zugreifen – dann sieht die Anweisung für die Datenbankverbindung so aus:

```
$conn = new mysqli ("127.0.0.1", "dbuser", "geheim", "musik");
```

Die Variable `$conn` ist ein Verbindungsobjekt, über dessen Methoden und Eigenschaften Sie MySQL-Datenbankoperationen vornehmen können.

Nun können Sie die `mysqli`-Methode `query()` verwenden, um SQL-Abfragen an die Datenbank zu senden. Das folgende Beispiel erstellt einen neuen Eintrag in der Tabelle `cds`:

→

```
$success = $conn->query ("INSERT INTO cds
     (cd_interpret, cd_titel, cd_jagr, cd_songs)
     VALUES (1, \"Ride The Lightning\", \"1984\", 8)");
```

Bei solchen Änderungsabfragen liefert die Eigenschaft `$conn->affected_rows` die Anzahl der geänderten Datensätze. Wenn Sie dagegen mit `SELECT` eine Auswahlabfrage durchführen, erhalten Sie ein Ergebnisobjekt:

```
$query = $conn->query
     ("SELECT int_name, cd_titel FROM interpreten, cds
      WHERE int_nr=cd_interpret");
```

Sie können `$query` benutzen, um die erhaltenen Datensätze auszulesen. PHP bietet zu diesem Zweck die Methode `fetch_row()`, die jeweils den nächsten Datensatz zurückliefert. Dies lässt sich am praktischsten in einer Schleife der folgenden Bauart nutzen:

```
echo ("<table border=\"1\">\n");
echo ("<th>Interpret</th><th>CD</th>\n");
while (list ($name, $titel) =
     $query->fetch_row ()) {
   echo ("<tr><td>$name</td><td>$titel</td></tr>\n");
}
echo ("</table>\n");
```

Interessant ist hier das Konstrukt `list($var1, $var2, ...)`: Sie können es auf der linken Seite einer Wertzuweisung benutzen, um den Einzelvariablen in der Liste nacheinander die Werte eines Arrays oder einer anderen Liste zuzuweisen. Auf diese Weise können Sie es sich leicht ersparen, eine Array-Variable zu definieren und deren Felder wieder auszulesen.

Entwurf der Datenbanktabelle

Die Datenbanktabelle für das Gästebuch soll folgende Felder enthalten: Name (Feldname `gb_user`), E-Mail-Adresse (`gb_mail`), Instrument (`gb_instrument`), Titel (`gb_subject`) und Nachricht (`gb_message`). Außer der Nachricht selbst sind alle Angaben optional. Zusätzlich wird ein Feld namens `gb_id` eingeführt, das als Primärschlüssel dient; es wird per `AUTO_INCREMENT` automatisch durchnummeriert. Die meisten Felder besitzen den Datentyp `VARCHAR` (Text mit angegebener Maximallänge). Die einzigen Ausnahmen sind `gb_id` (`INT`) sowie `gb_message` vom Typ `TEXT` (Textblock mit bis zu 65.536 Zeichen).

Bevor die Gästebuch-Anwendung in Betrieb genommen werden kann, muss diese Datenbank eingerichtet werden. Starten Sie dazu das Kommandozeilenprogramm `mysql`, und geben Sie folgende Anweisungen ein:

```
mysql> CREATE DATABASE guestbook;
Query OK, 1 row affected (0.00 sec)
mysql> use guestbook
```

```
Database changed
mysql> CREATE TABLE guest
    -> (gb_id INT PRIMARY KEY AUTO_INCREMENT, gb_user VARCHAR(30),
    ->  gb_mail VARCHAR(30), gb_instrument VARCHAR(50),
    ->  gb_subject VARCHAR(50), gb_message TEXT);
Query OK, 0 rows affected (0.04 sec)
```

Vorhandene Einträge anzeigen

Wenn Sie sich in der Datei *guest_start.fla* umschauen, werden Sie auf der Ebene *labels* die beiden Markierungen *read* und *write* bemerken, die dem Lesen vorhandener beziehungsweise dem Hinzufügen neuer Einträge dienen. Wählen Sie das Schlüsselbild auf der Ebene *gui* im Bereich *read* aus; hier müssen die Textfelder zunächst mit Variablen verknüpft werden. Markieren Sie sie dazu nacheinander, und tragen Sie unter *Var.* in der Eigenschaftenleiste folgende Variablennamen ein: `user`, `mail`, `instrument`, `subject` sowie `message`. Es handelt sich um dynamische Textfelder; das Laden der Variablen aus der entsprechenden PHP-Anwendung sorgt automatisch dafür, dass die einzelnen Einträge in den Feldern erscheinen.

Weisen Sie dem Schlüsselbild in Bild 2 auf der Ebene *actions* das folgende Skript zu:

```
// getData() liest den aktuellen Eintrag
function getData ()
{
    loadVariablesNum ("guestdata.php", 0, "POST");
}

// Aktueller Eintrag: Nr. 1
var eintrag = 1;
getData ();
stop();
```

Die Funktion `getData()` ruft `loadVariablesNum()` auf und beauftragt so das PHP-Skript *guestdata.php*, die Daten des aktuellen Eintrags zu liefern. Durch die Angabe der Methode `"POST"` wird sichergestellt, dass der Flash-Film auch seine eigenen Variablen an die PHP-Anwendung sendet – diese erhält dadurch den Wert der Variablen `eintrag` und weiß, welcher Datensatz angefordert wird.

Unter der Funktionsdefinition wird `eintrag` auf den Anfangswert 1 gesetzt. Der anschließende Aufruf liest also den ersten Datensatz. Zu guter Letzt sorgt `stop()` dafür, dass der Film an dieser Stelle anhält.

Als Nächstes erhält auch der linke Blätter-Button ein Skript:

```
on (release) {
   eintrag--;
   getData();
}
```

Der Wert der globalen Variablen `eintrag` wird also um 1 vermindert. Der anschließende Neuaufruf von `getData()` stellt sicher, dass der entsprechende Datensatz gela-

den wird. Wie Sie sehen, kümmert sich dieses Skript nicht um die Unterschreitung des Grenzwertes 1 – es ist praktischer, die entsprechende Logik in das PHP-Skript *guestdata.php* einzubauen.

Der rechte Pfeil benötigt natürlich ein beinahe identisches Skript; der einzige Unterschied ist, dass `eintrag` hier um 1 erhöht wird:

```
on (release) {
   eintrag++;
   getData();
}
```

Schließlich braucht auch die Schaltfläche *Eintragen* noch ein Skript – sie springt zum Frame *write*, damit ein Benutzer sich ins Gästebuch eintragen kann:

```
on (release) {
    gotoAndStop ("write");
}
```

Das Wichtigste für diese Anwendung sind allerdings nicht diese kurzen ActionScript-Anweisungsblöcke, sondern das PHP-Skript. Geben Sie in einen Editor Ihrer Wahl folgenden Code ein, und speichern Sie ihn unter dem Namen *guestdata.php* im gewünschten Verzeichnis Ihrer Website:

```
<?php

   // ... Function cgiParam() kopieren oder einbetten ...

   // Eintragsnummer vom Flash-Film übernehmen; Standardwert 1
   $eintrag = cgiParam ("eintrag", 1);

   // Verbindung zur MySQL-Datenbank herstellen - anpassen!
   $conn = new mysqli ("127.0.0.1", "dbuser", "geheim", "guestbook");

   // Anzahl der Einträge ermitteln
   $query = $conn->query ("SELECT COUNT(gb_id) FROM guest");
   $maxEintraege = 1;
   if ($query->num_rows > 0) {
      list ($maxEintraege) = $query->fetch_row ();
   }

   // $eintrag anpassen, falls zu groß oder zu klein
   if ($eintrag < 1) {
      $eintrag = 1;
   }
   if ($eintrag > $maxEintraege) {
      $eintrag = $maxEintraege;
   }

   // Eintrag aus der Datenbank lesen
   $query = $conn->query
      ("SELECT gb_user, gb_user, gb_instrument, gb_subject,
        gb_message FROM guest WHERE gb_id=$eintrag");
```

```
    list ($user, $mail, $instrument, $subject, $message)
          = $query->fetch_row ($result);

    // Datentyp "nur Text" setzen
    header ("Content-type: text/plain");

    // Alle Variablen URL-codieren
    $user = urlencode ($user);
    $mail = urlencode ($mail);
    $instrument = urlencode ($instrument);
    $subject = urlencode ($subject);
    $message = urlencode ($message);

    // Ausgabe an den Flash-Film
    echo ("user=$user&mail=$mail&instrument=$instrument&subject=$subject&message=$me
ssage&eintrag=$eintrag");

?>
```

Neue Einträge vornehmen

Unter dem Schlüsselbild *write* finden Sie den Teil des Films, der dem Erstellen eines neuen Gästebucheintrags dient. Der Bildschirmaufbau ist fast identisch mit dem Bild *read*; allerdings werden hier Texteingabefelder statt der dynamischen Textfelder verwendet. Weisen Sie ihnen ähnliche Variablennamen zu wie den Feldern im vorigen Unterabschnitt (der Unterschied ist das vorangestellte `e_` für »Eingabe«): `e_user`, `e_mail`, `e_instrument`, `e_subject` und `e_message`).

Weisen Sie dem Schlüsselbild auf der Ebene *actions* das folgende Skript zu, um die Textfelder bei einem erneuten Eintrag wieder zu leeren:

```
e_user = "";
e_mail = "";
e_instrument = "";
e_subject = "";
e_message = "";

stop();
```

Nun benötigt die Schaltfläche *Speichern* folgendes Skript:

```
on (release) {
    loadVariablesNum ("guestsave.php", 0, "POST");
    gotoAndStop ("read");
}
```

Die zweite Schaltfläche, *Abbrechen*, soll dagegen unverrichteter Dinge zum Lesen zurückkehren:

```
on (release) {
    gotoAndStop ("read");
}
```

Zu guter Letzt benötigen Sie noch das PHP-Skript *guestsave.php*, das durch einen Klick auf *Speichern* aktiviert wird und den neuen Eintrag in der Datenbank ablegt:

```
<?php

   // ... Function cgiParam() kopieren oder einbetten ...

   // Formulardaten auslesen
   $user = cgiParam ("e_user", "");
   $mail = cgiParam ("e_mail", "");
   $instrument = cgiParam ("e_instrument", "");
   $subject = cgiParam ("e_subject", "");
   $message = cgiParam ("e_message", "");

   // Datenbankverbindung herstellen
   $conn = new mysqli ("127.0.0.1", "dbuser", "geheim", "guestbook");

   // Wurde zumindest eine Nachricht geschrieben?

   if ($message) {
      // Eintrag vornehmen
      $success = $conn->query ("INSERT INTO guest
           (gb_user, gb_mail, gb_instrument, gb_subject,
            gb_message) VALUES (\"$user\", \"$mail\",
            \"$instrument\", \"$subject\", \"$message\")");
   }

   // Neue Eintragsanzahl ermitteln
   $eintrag = 1;
   $query = $conn->query ("SELECT COUNT(gb_id) FROM GUEST");
   if ($query->num_rows > 0) {
      list ($eintrag) = $query->fetch_row ();
   }

   // Datentyp "nur Text" festlegen
   header ("Content-type: text/plain");

   $user = urlencode ($user);
   $mail = urlencode ($mail);
   $instrument = urlencode ($instrument);
   $subject = urlencode ($subject);
   $message = urlencode ($message);

   echo ("user=$user&mail=$mail&instrument=$instrument&subject=$subject&message=$me
ssage&eintrag=$eintrag");

?>
```

In Abbildung 10-4 sehen Sie das Gästebuch in Aktion, und zwar beim Eintragen eines neuen Datensatzes.

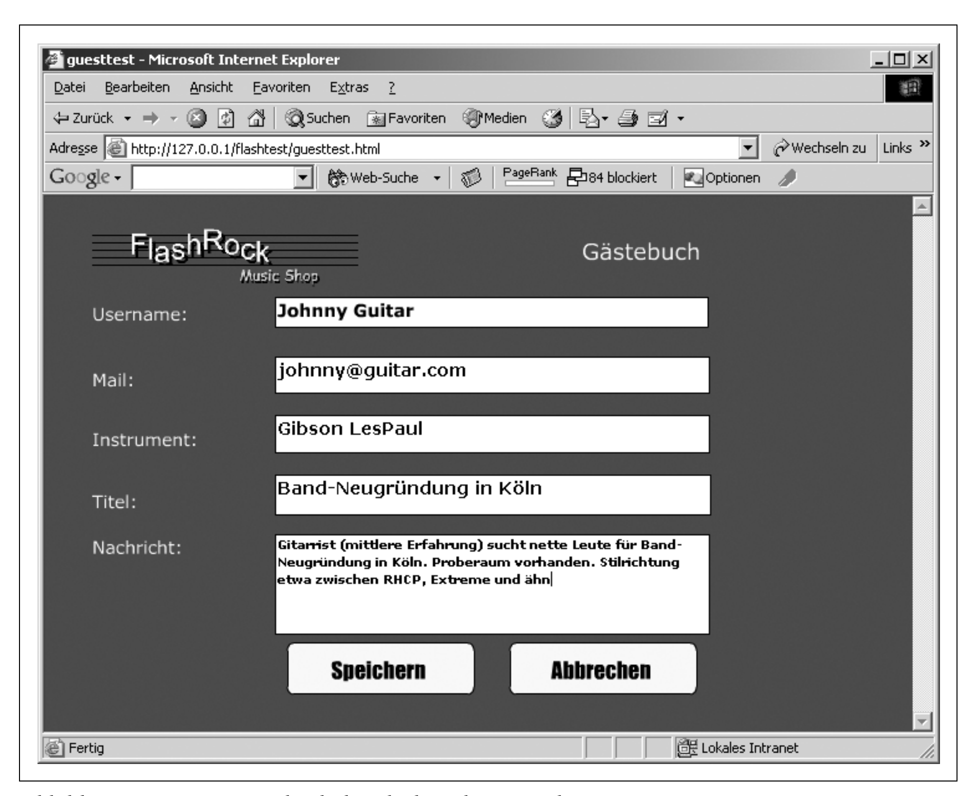


Abbildung 10-4: Das Gästebuch des FlashRock Music Shop